NTNU
Norwegian University of Science and Technology

EEG-based Brain Computer Interface for Controlling a Drone

using machine learning, optimized for a battery powered device

Kristian Husevåg Krohn
Adrian Ribe

Master of Science in Electronics

Submission date:	June 2018
Supervisor:	Per Gunnar Kjeldsberg, IES
Co-supervisor:	Marta Molinas, ITK
	Lars Magne Lundheim, IES

Norwegian University of Science and Technology
Department of Electronic Systems

Abstract

Brain Computer Interfaces enable the use of brain waves to control computer-based external devices. It can, however, be difficult to trigger a specific brain wave, and it might not be repetitive enough to be used in such a manner. EEG is used to record brain waves, where the parts of the recorded signal that is related to eye movements are normally considered to be artifacts (undesired). In this work, however, EEG recordings of eye movements are used to create datasets for training a machine learning algorithm, used for controlling a drone in real time.

A list of features is presented through literature search and comparison of eye movement plots. A non-linear SVM classifier and a greedy feature selection algorithm is used to distinguish between blinks, looking straight ahead, to the left, right, up and down. Data collected from two subjects is presented, obtaining an average accuracy of 94% offline when training on both subjects. A state-machine was designed for controlling the drone and evaluating online classification. The down movement was discarded from the state machine due to observed degradation in online performance when including this class, online classification therefore only uses 5 eye movements. An average online accuracy of 94.5% was obtained when training on both subjects and testing separately.

With proof of concept, the selected features and choice of algorithm is optimized for an implementation on a battery powered device with respect to energy efficiency and classifier accuracy. Execution time speedup is used as a measure for energy efficiency in this thesis, as this can be combined with a race-to-halt strategy to optimize for energy efficiency. A brute force search is utilized (together with the greedy feature selection algorithm) to find the best feature vector, presenting 9 features with a total extraction time of 242 μs on a personal computer. Through this method, a speedup of 48x was achieved with a decrease of 2.1% in online accuracy when training and testing on Subject 1. The subjects are able to use the system to control a drone in real time, showing that eye movements can be used for devices demanding quick response. The system was observed to be delicate, demanding the subjects to stay very concentrated and for the electrodes to be positioned correctly, with good skin contact.

Sammendrag

I denne masteroppgaven presenteres et system der prosesserte EEG-signaler fra et kommersielt produkt med åpen kildekode, blir brukt til å klassifisere mellom øyebevegelsene opp, ned, høyre, venstre, blunk og å se rett frem ved bruk av maskinlæring. Flere metoder blir presentert for å finne gode egenskaper i disse dataene og for å finne en maskinlæringsstruktur som kan klassifisere bevegelsene. Utgangssignalene fra klassifikatoren brukes til å kontrollere en drone.

Data ble samlet inn fra to deltakere, der et brukergrensesnitt med en sirkel som beveger seg i forskjellige retninger ble designet for å gi instruksjoner til deltakerne i form av retning og tidspunkt for bevegelse. I teorien har det presenterte systemet for innsamling av data mulighet til å lagre data for 120 bevegelser i hver retning i løpet av en time (480 for blunk og rett frem). I praksis ble innhenting av data gjort i kortere intervaller slik at deltakerne kunne opprettholde denne hastigheten uten å miste fokus og bli slitne. Det ble laget et datasett til hver av deltakerne der antall lagrede bevegelser for hver av deltakerne var på henholdsvis 2415 og 2980 bevegelser.

For å gjøre klassifiseringen bedre, blir det foreslått ulike måter å abstrahere egenskaper ved dataen på, basert på litteratursøk av lignende oppgaver og inspeksjon av dataen som ble laget. Validering av disse egenskapene blir gjort gjennom selve klassifiseringen, der gode resultater tilsier at egenskapene er gode til å beskrive forskjeller mellom klassene. Litteratursøk kombinert med en eksperimentell fremgangsmåte ble brukt for å finne riktig maskinlæringsalgoritme, der et verktøy kalt Scikit-learn med mange ferdige implementasjoner av kjente maskinlæringsalgoritmer ble brukt. Flere av disse ble lært opp og testet på datasettene der forskjellige kombinasjoner av egenskaper fra dataen ble brukt av algoritmene. Isteden for å manuelt vurdere egenskapene, ble en grådig algoritme for utvalg av slike egenskaper benyttet. En grådig algoritme er en algoritme som til en hver tid velger det beste alternativet. Med hjelp av denne, og en eksperimentell fremgangsmåte, ble det oppnådd en treffsikkerhet på 94%, når data fra begge deltakerne ble slått sammen for å lære opp maskinlæringsalgoritmen. Dette var testet på data som var laget på forhånd og lagret på datamaskinen.

Å bruke en klassifikator i sanntid til å styre en drone, krever også en kontroller til å tolke klassifiseringene. Til å tolke disse ble det laget en tilstandsmaskin, som endrer tilstand basert på sekvensen til utgangssignalet til klassifikatoren. Denne tilstandsmaskinen blir også brukt til å evaluere hvor god klassifikatoren er når den prøver å klassifisere data som blir tatt opp i sanntid. Evalueringen ble gjort i et opptaksmiljø som kunne ta opp tilstanden til tilstandsmaskinen gjennom utskrifter i et terminalvindu og samtidig ta opptak av øynene til deltakerne via et webkamera. Det ble gjennom denne evalueringen observert at "ned" bevegelsene forstyrret klassifikatoren, og dermed forverret systemets evne til å klassifisere resten av retningene. Denne "ned" bevegelsen ble derfor ekskludert fra tilstandsmaskinen, som gjør at kun de 5 resterende bevegelsene ble brukt til sanntidsklassifisering.

Gjennom perioder der deltakerne testet klassifikatoren i sanntid, ble det observert at systemet er veldig delikat i forhold til konsentrasjon og hvor god kontakt utstyret har med huden. Når konsentrasjonen og kontakten var tilfredsstillende, ble det observert resultater med en treffsikkerhet på 94.5% ble observert når begge deltakerne lærte opp samme klassifikator.

For å optimalisere løsningen i forhold til energieffektivitet ble kjøretid brukt som en indikator på energibruk. Det viste seg at den grådige algoritmen valgte ut de mest krevende egenskapene. En optimalisering ble gjort, der antallet egenskaper ble redusert og den endelige kombinasjonen skulle være av de egenskapene som hadde kortest utregningstid med en definert nedre grense for treffsikkerhet for klassifikatoren. Denne kombinasjonen ble funnet med et komplett søk av alle egenskapskombinasjoner der antallet egenskaper var satt til 9 av 26 mulige egenskaper. En sammenlikning på deltaker 1 med egenskapskombinasjonen valgt av den grådige algoritmen og det komplette søket viste at utregningen av egenskaper og klassifisering fikk en hastighetsøkning på 48x med en reduksjon i treffsikkerhet på 2.1%. Med denne hastighetsøkningen kan energiforbruket minkes ved å minke klokkefrekvensen og spenningen.

Gjennom resultatene blir det konkludert med at fremgangsmåten for å finne riktige egenskaper, maskinlæringsstruktur, design av kontroller og energioptimalisering er velegnet for styring av drone med øyebevegelser. En video av en av deltakerne som flyr drone med øyebevegelser kan bli funnet her: https://www.youtube.com/watch?v=8S9dgh5TH0A

Preface

This is a master's thesis written at NTNU as part of the 2-year study programs Electronics - Embedded Systems and Electronics - Design of Digital Systems. The thesis was written during the spring semester of 2018 in cooperation with the Department of Industrial Cybernetics and Department of Electronic Systems. The idea of the project was brought up by our supervisor Professor Marta Molinas as a specialization project in the autumn semester of 2017 and was then continued for this master's thesis.

Trondheim, 6/6-18

Adrian Ribe & Kristian Husevåg Krohn

Acknowledgment

We would like to thank the following persons for their great help during the project. Our supervisors Lars Magne Lundheim, Per Gunnar Kjeldsberg and Marta Molinas for outstanding support, tips and guidance whenever needed. Orakeltjenesten for guidance on how to use the school servers for training machine learning models and the community at Stack Overflow for technical support. At last we would like to thank our friends and family for help and support along the way.

Adrian Ribe & Kristian Husevåg Krohn

Contents

Acronyms

ADC	Analog-to-Digital converter. 25, 28, 29
API	Application Programming Interface. 23, 25, 29, 95, 101
AR	Auto Regression. 20, 137, 138
BCI	Brain Computer Interface. 1, 2, 10–12, 25, 28, 51
CPU	Central Processing Unit. 126
CSP	Common Spatial Pattern. 21, 138
CV	Cross-validation. 62, 73, 78, 81, 85, 86, 88, 89, 91–93, 102, 103, 132, 146
DFT	Discrete Fourier Transform. 30
DVFS	Dynamic Voltage Frequency Scaling. 126, 144
EEG	Electroencephalography. I, 1–4, 6–8, 10–12, 18–20, 23, 25, 29, 35, 50, 51, 56, 59, 74, 103, 143, 145, 147
EOG	Electrooculography. 18, 21, 50, 53, 122, 145, 147
FEF	Frontal Eye Field. 9, 31, 34
FFT	Fast Fourier Transform. 20, 129, 162
FN	False Negative. 67
FP	False Positive. 67

FPGA Field-programmable gate array. 21, 147

FPS Frames Per Second. 102

GUI Graphical User Interface. 20, 21, 27, 36–38, 47, 144, 145

HFD Higuchi Fractal Dimension. 51, 52, 59, 130, 162

ICA Independent Component Analysis. 19, 21, 138

IIR Infinite impulse response. 35

IoT Internet of Things. 12

KDE Kernel Density Estimation. 133

KNN K-Nearest Neighbors. 19, 63, 64, 79, 80, 91, 92, 122

LinearSVC Linear Support Vector Classifier. 89, 90, 92, 94, 101, 105, 107, 108, 110–114, 121, 122, 127, 128, 138

OBS Open Broadcaster Software. 102

OS Operating System. 101

PFD Petrosian Fractal Dimension. 52, 163

PSE Power Spectral Entropy. 59, 162

PSI Power Spectral Intensity. 162

PTP Peak-to-Peak. 54, 55

PWM Pulse Width Modulation. 19

RBF Radial Basis Function. 75, 77, 84, 86–89, 92–94, 101, 105, 106, 114–123, 127, 128, 131, 136

RFE Recursive Feature Elimination. 61

RFECV Recursive Feature Elimination with cross-validation. 60, 62, 83, 84, 86, 88, 89, 91, 105, 125, 127, 131–135, 137, 140, 144, 146, 197

RIR Relative Intensity Ratio. 162

RTH Race-to-Halt. 126

STD Standard Deviation. 52, 53

SVM Support Vector Machine. I, 2, 19, 21, 61–64, 74–78, 84–94, 96, 101, 105–107, 111, 114–123, 127, 128, 131, 136, 138

TBR Theta-Beta ratio. 20, 58, 59, 163

TN True Negative. 67

TP True Positive. 67

USB Universal Serial Bus. 29

Chapter 1

Introduction

The first ever human recording of electrical activity in the brain using Electroencephalography (EEG) was reported by Hans Berger in 1929 [1]. EEG studies has since then been performed numerous times, including studies on head injuries, sleep disorders, strokes and epilepsy, among others. EEG studies used to be exclusive to the medical and scientific community as equipment used to measure EEG signals was expensive. Such equipment has recently been made available to a larger audience by becoming cheaper and more mobile and developer-friendly.

Another way of looking at EEG recordings is to study the possibility of using these signals for actuation. If one were able to undergo study and training to produce and decode specific repeatable signals, it would be possible for people to control computer-based technology. These signals are recorded with scalp electrodes, making this a non-invasive technique. The usage of scalp electrodes introduces a challenge in signal processing as undesired noise and interference from other sources is present in the measurements. If this challenge can be overcome, a Brain Computer Interface (BCI) that enables interaction between brain activity and computer technology can be used in real-time. This can have unlimited applications. Electric wheelchairs, prosthetic limbs and assisting robots among many other options, can be controlled to increase the life quality of people with physical impairments [2].

Artificial intelligence is currently a very popular field, where machine learning has been noticed to have the potential to solve a large number of problems. "Machine learning is concerned with algorithmically finding patterns and relationships in data, and using these to perform tasks such as classification and prediction in various domains" [3]. This technology has the possibility to find patterns in data that is difficult to find for a human. Using it on EEG data can provide results that are hard to accomplish by having a person visually inspect EEG recordings. Machine learning allows for problems to be solved without the need of a human to specify rules for every possibility when solving a problem. Instead, the algorithms take in data and learns from it through "experience". When using machine learning to perform classification, a machine learning model is created through this experience and is considered a classifier [3]. There are many types of classifiers, where some of the most popular are linear classifiers, Support Vector Machine (SVM), Decision Trees, Logistic Regression and Neural Networks [4].

An aspect of BCI, is mobility and the possibility of not requiring proximity to an electrical outlet to use the system. In this work, BCI is used to control a drone. And when controlling a drone, especially if it is flown outside, it is considered an advantage to be mobile. Mobile devices are generally powered by batteries and therefore has a limited supply of power. When designing systems intended for mobile devices, energy efficiency is an important aspect and needs to be considered to increase battery life.

Problem Description

The objective of this work is to control a drone with eye movements using EEG and is a continuation of previous work in TFE4520 Digital System Design, Specialization Project [5]. The task includes creating a dataset of EEG recordings while test subjects performs a set of eye movements, and to use machine learning classification to differentiate between the movements. Through proof of concept, the classification should be implemented in real-time allowing the classifier to be used as part of a real-world control application. The final system has an end goal beyond this thesis to be implemented on a portable battery powered device. Thus, energy efficiency is a key component, which should be considered and argued for. The classification process should be optimized with respect to energy efficiency and classifier accuracy.

A block diagram showing a simplified version of the complete system as used in real time can be seen in Figure 1.1.

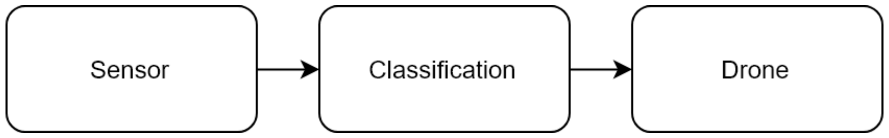


Figure 1.1: Simplified system overview

1.1 Objectives

The main objectives of this Master's thesis are:

1. Make a system for sufficiently efficient generation of eye movement EEG data
2. Make datasets for at least two test subjects
3. Find features of the data that are fit to distinguish eye movements
4. Find a machine learning structure that, combined with these features, can distinguish eye movements
5. Implement the classification in real-time
6. Design a drone controller based on the real-time predictions from the classifier
7. Optimize the machine learning structure and selected features based on accuracy and energy efficiency

1.2 Limitations

There are some limitations to this thesis. First of all, the number of test subjects is low and should ideally be a lot higher. Having few test subjects means that the argument for a cross-subject portability not necessarily holds up. Ideally, the classifier would be trained on a large group of subjects and tested on a group of subjects who were not used for training.

The datasets are created as part of this work, which can be considered a limitation as the quality of the dataset is subject to the persons evaluating them. The subjects creating the datasets can make voluntary or involuntary movements, which disturbs the EEG recordings, in addition to the fact that the quality of contact between electrode and skin can vary. A method to discard data was utilized to remove datapoints where disturbances were present in the signals. The method used to discard data was to visually inspect every datapoint in both datasets, where each author had responsibility for one dataset each. This means that there can be differences in the quality of the datapoints, due to the fact that the interpretation can have some variations across the inspectors.

Commercial grade equipment was used to record the EEG signals. It was observed that the placement of electrodes and contact with the skin affected the quality of recordings, therefore also affecting the quality of classification. The system is also dependent on the subjects to stay very focused during recording of data and flight. This could also affect the quality of datapoints in the datasets as the gathering of data was performed in intervals, allowing some variance in electrode placement and subject concentration.

Feature selection and choice of machine learning algorithm is based on energy efficiency and classifier accuracy. When designing other parts of the system, such as the pre-processing and controller, energy efficiency has been kept in mind but was not the main focus. This allows the system to be further optimized for a battery powered implementation.

The number of machine learning algorithms and structures tested is limited. The machine learning toolkit used in this work is Scikit-learn [6], which provides high-level implementation of many popular machine learning algorithms. The support for creation of neural networks is however low and the field of deep learning was unexplored by the authors. It is possible that a neural network could be created through another toolkit providing support for energy efficient machine learning, such as TensorFlow Lite, to create a more energy efficient classifier. This would however demand more time and would not be guaranteed to provide as good results as presented in this thesis within the same time period.

The energy optimized feature vector was only tested on Subject 1. The same procedure for obtaining that feature vector should ideally be applied to Subject 2, and when training a model on both subjects, to get a proper evaluation of the possible energy savings.

1.3 Approach

Nearly all objectives were approached with analysis, hypothesis, experimentation and observations, providing insight and feedback along the way.

To create a method for generating datasets, literature on similar projects to this thesis has been used. Looking at what others do and learn from their success and mistakes provided a good basis of a good data generation method. When proposing the use of different features, which features to select and what machine learning algorithms to use, the same basis with literature search was used in addition to an experimental approach with the different classifiers to get a feel of how good the selected features and algorithms are. Having a decent understanding of how the features and algorithms are used before experimenting is started, provided a good intuition of what works well and why.

To evaluate the performance of the classification, standard metrics used in machine learning literature were used throughout the thesis, both for offline and real time testing. A state machine was designed for the use of classifier outputs to control a drone, where an experimental approach with analysis, hypothesis, experimentation and observations was used. When it comes to energy efficiency, a comparison of different algorithms and selection of features using execution time as an energy efficiency indicator was used. Only the energy usage to actually classify a movement (not train the algorithm) was of interest, so the execution time for classifying a movement should be compared across several machine learning algorithms. Calculating the execution time for each of the presented features and comparing them, as well as testing different methods of feature selection was performed to show the possible energy efficiency optimization.

1.4 Structure of the report

This report is organized into eight chapters. An overview of the main background theory and a literature survey is presented in Chapter 2. Chapter 3 presents the system design with an overview, excluding the vital parts of classification, complexity, controller and energy efficiency. Chapter 4 will to some degree go in depth into machine learning, features and performance metrics where Chapter 5 presents the offline classification results. A real time implementation, design of drone controller and real time classification results is presented in Chapter 6. The real time classification is then optimized for energy efficiency in Chapter 7, where an evaluation of the overall system concludes the chapter. Chapter 8 is the last chapter and will evaluate if the objectives in this thesis are met, conclude and give recommendation for future work.

1.5 Main contributions

- Design of a system allowing a user to control a drone using eye movements, using methods based on literature search, experimentation and observation.

- Proposal of a list of features that was proven to be useful for eye movement classification of EEG signals.

- Creating a technique for selection of features by utilizing a popular greedy algorithm and a brute force search to find the best "Low-power" combination of features. A fitting number of features was found through observation of classifier performance during the feature selection procedure of the greedy algorithm. Through the use of this method, it was possible to find a feature vector that gave a speedup of 48x with a decrease of 2.1% classifier accuracy, compared to the feature vector found by the greedy algorithm.

- Creating an evaluation method for online classification performance for eye movements by utilizing a web camera and prints in a terminal. By looking at the eye movements and the state of the machine through prints at the same time, it was possible to validate the movements and evaluate the online performance.

- Presentation of many machine learning concepts and methods in a simplified matter, so that a reader without competence in machine learning can understand these concepts and methods.

Chapter 2

Background Material

This chapter gives a short overview of what makes it possible to measure brain activity, what measurements are used in this thesis, how these measured signals differ from each other and an introduction to machine learning. The presented material is from a literature search performed to get an understanding of the underlying processes utilized in this thesis.

2.1 Electroencephalography and brain anatomy

The Electroencephalography (EEG) is a technique to record the oscillations of the brain's electrical potentials measured from electrodes placed on the human scalp [7]. A human brain can be looked at in three parts, the brain stem, cerebellum and the cerebrum as shown in Figure 2.1.

The brain stem is the infrastructure where nerve fibers transmit signals between the spinal cord and brain centers. The cerebellum on the other hand has been associated with fine control of muscle movements but has also been shown to play a role in cognition [7].

The cerebrum is divided into two almost equal halves, the left and right cerebral hemisphere. The outer portion of the cerebrum, which is called the cerebral cortex, contains around 100 billion neurons and is believed to produce most of the electrical potentials in the scalp [7].

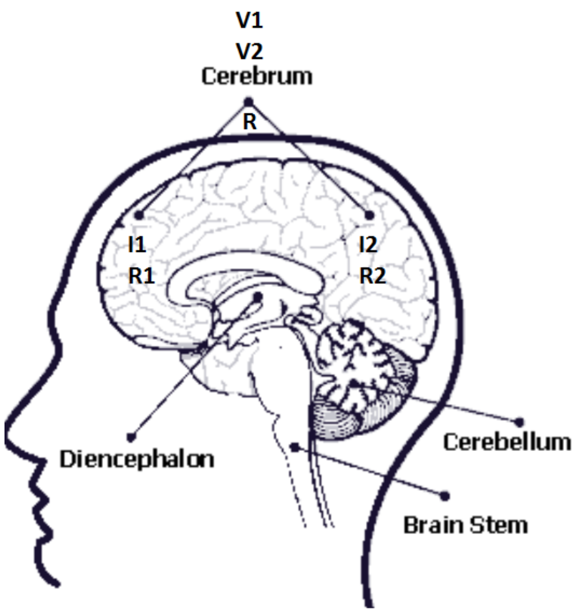


Figure 2.1: Overview of the human brain. The superposition principle says that if we turn on all the current sources ($\mathbf{I}_1....\mathbf{I}_n$) at the same time, the potential at location **R** is the linear sum of individual potentials ($\mathbf{V}_1....\mathbf{V}_n$). [8]

Scalp recordings do not depend much on electrode size, as scalp potentials are space-averaged by volume conduction between brain and scalp. A single electrode used on the scalp may cover tissue masses containing roughly 100 million - 1 billion neurons, where each neuron can contain about 10 - 100 thousand synapses [7]. Electrical activity in multiple nearby neurons adds up to local field potentials (local sources). If enough local sources synchronize, the voltage potential adds up to having a large enough amplitude to be read on the scalp. Figure 2.1 above illustrates how superposition is used with EEG.

2.2 Frontal cortex

The frontal cortex is responsible for execution of all forms of action. Within this area is the prefrontal cortex, which has the general function of temporal organization of actions toward biological or cognitive goals [9]. This includes eye movement, emotional behavior, intellectual performance, speech, etc. [10]. In this work the aim is to differentiate between different eye movements. The prefrontal cortex, marked in orange in Figure 2.2 therefore ends up being the area of focus.

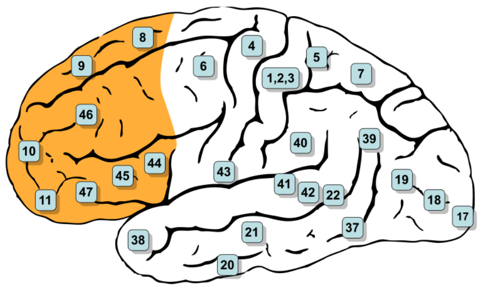


Figure 2.2: Figure of the left cerebral hemisphere where the prefrontal cortex is marked in orange. This is a recreation of the original picture from [11].

Figure 2.2 is called Brodmann's brain map [12], from 1909. To this day it forms the basis for "localization" of functions in the cerebral cortex. Brodmann's "areas" are still used to designate cortical functional regions, such as Area 4 for motor cortex, Area 17 for visual cortex, and so on [13]. The part of highest interest in this work is indexed as the number 8, and is named the Frontal Eye Field (FEF). In humans, functional imaging studies have demonstrated increased FEF activation during all visually guided quick simultaneous movement of both eyes, reflex or voluntary [14].

2.3 Electrical activities

The electrical activities among neurons can be divided into two major categories: spontaneous potentials and event related potentials [7]. Spontaneous potentials are always present and can for instance be electrical activity as a result of sleep cycles. While event related potentials are the changes in the electrical activity as a direct reaction to an event, such as moving an arm or some internal or external stimuli. It is important to note that the event related potential might give different measurements, depending on what state the mind is in [7].

The electrical activities originating from the brain can produce five major brain waves which are classified by their frequency ranges. These are known as Alpha, Beta, Delta, Gamma and Theta waves [15]. Table 2.1 shows the different frequency ranges for the different brain waves.

Table 2.1: Brain waves and frequency ranges [15]

Brain wave	Frequency range (Hz)
Delta	0.5 − 4
Theta	4 − 8
Alpha	8 − 13
Beta	13 − 30
Gamma	30 − 100

Artifacts

There are however other sources to electrode potentials in the EEG measurements that does not originate from the neuron- and synaptic activity. These are called artifacts and can originate from muscle activity or interference from electrical equipment, among many other sources [15]. Artifacts originating from biological activity in the brain are called *biological artifacts*, while artifacts from components such as electrical equipment is called *technical artifacts*. Eye movements and blinks are called *ocular artifacts*, and usually has higher amplitudes than brain signals [15]. While voltage potentials originating from brain signals are within the $0-100$ μV range, the ocular artifacts can vary from 10 μV - 5 mV [16, 15]. These artifacts originate from the electrical charge of the eye, where the cornea is positively charged and the retina is negatively charged, creating a dipole. When the eye or eyelid moves, the potential field from the charge changes, which is easily detected in the EEG measurements [17]. These changes in the EEG measurements, triggered by the eye movements and blinks are the signals that are used and classified in this work.

2.4 Brain-Computer Interface

Brain Computer Interface (BCI) technology can be looked at as an interface allowing communication between a wired brain and an external device [18]. BCI uses direct measurements from the brain activity using electrodes placed on the scalp, instead of using normal communication pathways, which depends on nerves and muscles [19]. Figure 2.3 shows the basic components of a typical BCI.

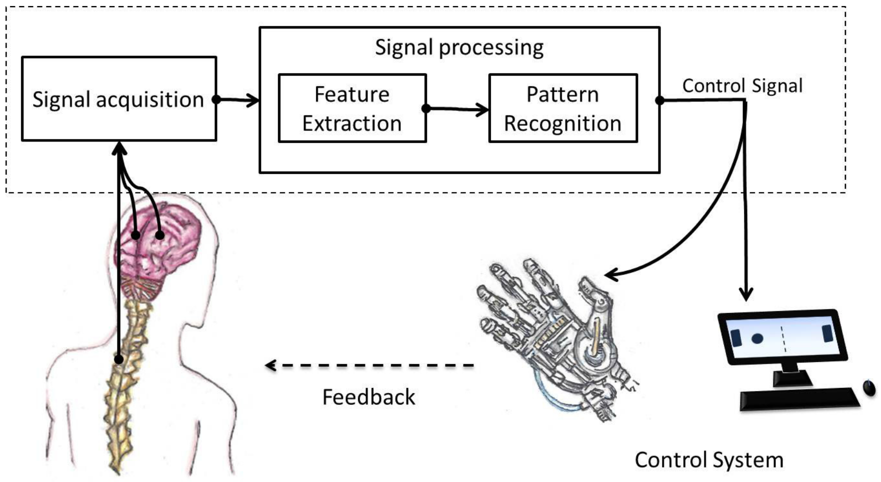


Figure 2.3: The figure illustrates the stages from input to output in a BCI. The input signal from the electrodes is often processed to better the signals acquired, to be able to translate the brain activity into some control signal for an external device [20].

BCI is a computer-based system which analyzes and utilizes the EEG signals acquired from the scalp, often to be studied or used to control some external device. In these systems, the user manipulates the brain activity to produce signals which can be used to control computers or communication devices [21]. There are several definitions of the BCI term, but it can be described by the following according to Cloyd [22]:

- BCI must record the electrical activity directly from the brain
- The device provides feedback for the user
- The feedback is provided in real time
- The system must rely on voluntary control

This means that BCI is a system which uses the brain activity to control a device where some type of feedback is given while the device is being controlled. Some real time feedback could be the visual feedback from for example a robot-arm movement, while controlling a robot in real time. Using the same example, BCI would require that the robot only moves when it is desired to by the user. There is a lot going on inside the brain that *could* trigger some event, in addition to the electrical activity that *should* trigger the event. Any unwanted activity that may trigger the robotic movement should be handled in some way, allowing for voluntary control.

BCI was initially developed as technology used with bio-medical applications, such as replacing lost motor function [23]. The technology has also expanded outside the medical appliances to entertainment, smart environment (Internet of Things), advertisement and education among others [19]. BCI has allowed to make new approaches to interacting with computers, as this technology has become more available for a larger audience through the introduction of cheaper EEG headsets. This has opened up for the production of consumer friendly BCI products which comes as an assembled device, ready to plug and play.

The introduction of EEG is not the only factor for the increase in BCI popularity for consumers [22]. The popularity has risen as a result of the popularity of the gaming industry, and a shift from consoles such as XBOX and Playstation to other devices such as smart phones [22]. This has created a cultural accept for computer-based technology and different ways to interact with that technology. Through introduction of new types of controlling methods in the ever-growing gaming industry, such as Microsoft's Kinect [24] and HoloLens [25], the interest for new and novel ways to interact with technology increases [22]. The rising interest in new types of controllers, the rising knowledge of brain activity and the introduction of cheaper electronics is what the authors believe has spiked the popularity of BCI over the last years. There are several BCI-devices available on the market, some of these are produced by companies such as Neurosky [26], Emotiv [27] and OpenBCI [28]. In this work, a BCI-system made by OpenBCI is used.

2.5 Classification with machine learning

This section provides an interpretation of the different steps in machine learning from different literature and is not based on any "new" findings in the work of this thesis. However, figures have been created and connected to some practical examples to better illustrate some of the concepts in hope of making it easier to understand.

Machine learning is a technique which gives a computer the ability to learn, without being explicitly programmed. Instead of making an application where everything is specified by a human, data is simply fed into a machine learning algorithm. There are two main types of learning methods: supervised- and unsupervised-learning [29].

Supervised learning depends on a supervisor to give the machine learning algorithm both input data and desired output. Unsupervised learning does not provide the desired output to the classifier, but lets the algorithm try to organize the input data on its own [30]. In this thesis, supervised learning is the learning method of focus.

For supervised learning, the machine learning algorithm will use the set of input data and desired output to train and find connections between input and output, this stage is called *training* [31]. The training stage realizes an inferred function called a *decision function* and a boundary called a *decision boundary*, which are used in the second stage called *testing*. The decision function is a function which maps input data to a scalar [32], so that:

$$f(x) = \begin{cases} \mathbb{R}_{>0} & : \text{if } x \in \text{Class 1} \\ \mathbb{R}_{\leq 0} & : \text{if } x \notin \text{Class 1} \end{cases}$$

The decision boundary is a boundary learned by the classifier, used to separate data in different classes from each other [32]. An example of a decision boundary is shown in Figure 2.5.

To better illustrate what the different terms means when connecting them to a specific machine learning problem, the rest of this section is divided into two examples. The machine learning terminology used in this thesis is introduced in the first example and is a machine learning problem where a classifier tries to distinguish between positive and negative numbers. In the second example a classifier tries to distinguish between different types of the Iris flower.

2.5.1 Example 1: Classify positive and negative numbers

Figure 2.4 shows a simplified illustration of a supervised learning implementation for the training stage.

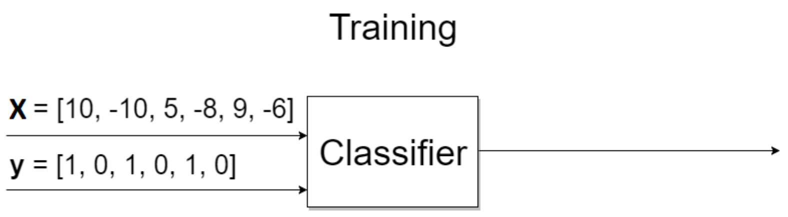


Figure 2.4: Illustration of how the training stage works. The classifier is fed two input objects **X** = [$\mathbf{x}_0$, $\mathbf{x}_1$...$\mathbf{x}_k$] and **y** = [y_0, y_1...y_k] with k datapoints. There is no output during the training stage

Data is essential to train a classifier, and the data is stored in the form of a *dataset*. Datasets used in supervised learning methods has a collection $\mathbf{X}$ of *datapoints* $\mathbf{x}_0$, $\mathbf{x}_1$...$\mathbf{x}_k$ and a collection $\mathbf{y}$ of *labels* y_0, y_1...y_k. The datapoints $\mathbf{x}_0$, $\mathbf{x}_1$...$\mathbf{x}_k$ are represented by so called *features*, which can be any representation of data. A feature could for example be direct sample-values from electrode recordings, the median of all those sampled values or any other attribute of the data [33]. A n dimensional vector containing all n features representing the data point is called a *feature vector* [3]. In this thesis, each datapoint is an eye movement, where the feature vector consists of features that represents the eye movement (datapoint). Therefore, when talking about datapoints, it is the eye movements and the number of them that is of focus. While the focus when talking about the feature vector, is how these eye movements are represented in the form of features. The labels y_0, y_1...y_k indicates which *class* a datapoint belongs to, and it is therefore often called the class label. It is important to note that there is no actual output during the training stage, but that the datapoints are labelled so that the classifier can realize the decision function and decision boundary.

Figure 2.5 shows an illustration of the ideal classification of the simple system introduced in Figure 2.4, where the feature vector is of dimension $n = 1$. This means that the datapoint is represented by a single feature, which in this case is the value of some integer.

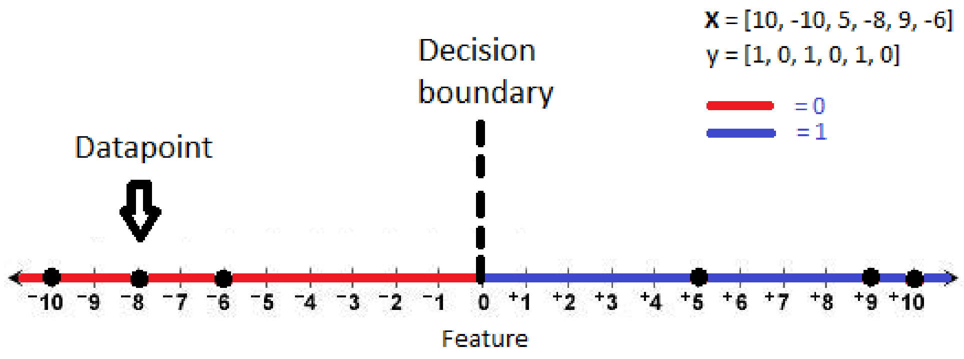


Figure 2.5: Illustration of a one dimensional feature vector classification.

There were in total 6 datapoints fed into the classifier, where the feature vector is of $n = 1$ dimension and the feature is a positive or negative integer. Looking at the value of the labels, it illustrates that negative numbers belong to class 0, and positive to class 1. As mentioned, the datapoints and labels are used to train the classifier, realizing the decision function and decision boundary. Thus, the more datapoints used for training the better and more accurate the classification of the new datapoint. The decision function and decision boundary can then be used to decide the class of a new datapoint not introduced in the training stage, this is called *testing*. The output of the classifier when testing is called a *prediction*. In the example in Figure 2.5, the decision boundary is a straight line at the number 0, dividing negative and positive numbers. The decision function classifies the new datapoint as class 1 if its scalar output is larger than 0, and as class 0 otherwise (as defined in Equation 2.5.) In a case such as this ideal example, which happens to have a linear decision boundary, a decision function can be as shown in Equation 2.1.

$$f(x) = x + b \tag{2.1}$$

The value b says something about the bias of the decision boundary and x is the datapoint that the classifier should predict. In this case, the decision boundary is a straight line at the number 0, so the bias value is $b = 0$. When training is over, one might start feeding new datapoints to the classifier. Some input could be:

$$\mathbf{X} = [7, -8, 3, -5, 11, -6]$$

The classifier is now in the testing stage and will try to put each datapoint $\mathbf{x}_0$, $\mathbf{x}_1 ... \mathbf{x}_k$ in class 0 or 1, where the feature vector is of dimension $n = 1$. By looking at the feature vectors and using their values as input in the decision function as defined by Equation 2.1, it should be possible to see what the classifier would predict.

Figure 2.6 shows a simplified illustration of the testing stage of a supervised learning implementation, with an ideal prediction accuracy of 100%.

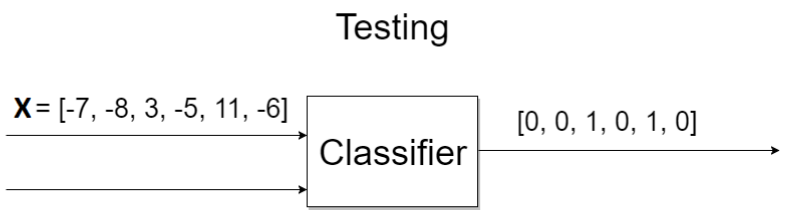


Figure 2.6: Illustration of the testing stage. New datapoints are fed into the classifier and the class the data belongs to is predicted by using the decision function.

The new datapoints are fed into the classifier where the realized decision function and decision boundary are used to predict the class of the data. It can only choose between the values of the labels given in the training stage, so it either outputs a 0 or a 1. Figure 2.6 shows that a list of predictions is output from the classifier. This is not the case when predicting one datapoint at a time but is a result of a dataset with several datapoints being put into the classifier.

2.5.2 Example 2: Classify types of Iris flower

To illustrate that the features and the dimension n of the feature vector can vary based on the classification task, one could look at the Iris dataset. The Iris dataset is a classic dataset used in machine learning [34]. The task is to discriminate between three types of the Iris flower, based on features such as sepal length and sepal width, among others. The Iris dataset has 150 datapoints and 4 features, where each datapoint is one of the types of Iris flower and the feature vector represents these flowers with the 4 features. Some of the feature vectors and class labels can be used in the training stage when choosing to use two of the features, sepal length (S_L) and sepal width (S_W), to classify the type of Iris flower:

$$\mathbf{X} = [[1.2, 0.2], [4.0, 1.3], [5.8, 1.9]]$$

$$\mathbf{y} = [0, 1, 2]$$

$$n = 2$$

Label 0 is the first Iris type called Setosa, Label 1 is the type called Versicolour and Label 2 is the type called Virginica. The collection of datapoints $\mathbf{X}$ consists of 3 datapoints with a feature vector dimension of $n = 2$. Figure 2.7 shows a plot of a classifier trained on the 3 feature vectors where the color blue represents class Label 0, light blue represents class Label 1 and red represents class Label 2. The illustration-method was inspired by [35].

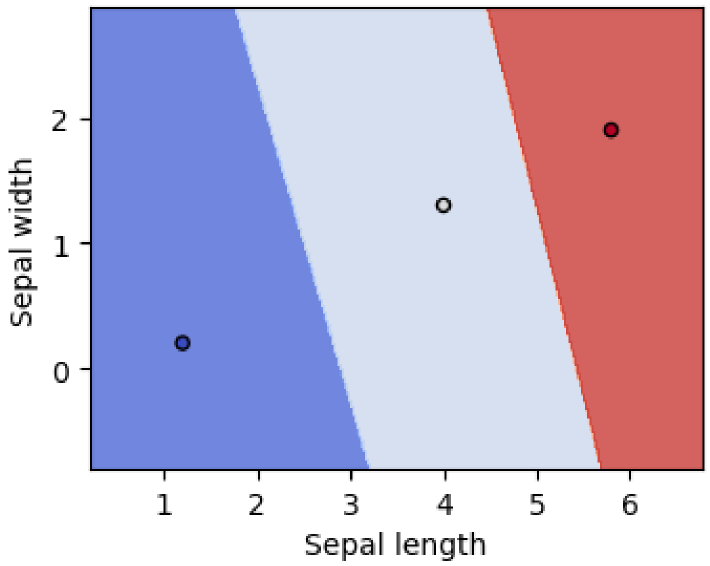


Figure 2.7: Illustration of Iris example in two-dimensional space, given by the features sepal length and sepal width [35]. Blue represents class Label 0, light blue represents Label 1 and red represents Label 2.

The datapoints are used to train a classifier to discriminate between the different classes by realizing the decision function and decision boundary. The decision boundaries are seen as the straight lines separating the different colors. Figure 2.7 shows the result of an actual implementation of a linear classifier trained on the dataset with **X** datapoints and **y** labels. If a new datapoint with the feature vector $x_3 = [1.0, 1.0]$ is introduced in the testing stage, it should be possible to visualize the prediction from the classifier by looking at the decision boundaries in Figure 2.7.

There are many different types of machine learning algorithms which can solve problems such as the classification examples introduced among many others [6]. Each algorithm works in a specific way to realize decision functions from datapoints given in the training stage in the best possible way.

2.5.3 Dataset and feature vector

As mentioned in the Section 2.5, the machine learning algorithm purely bases itself on the data it gets. Therefore, the algorithm will always be limited by the quality of the data. Low quality data, both for training and testing, will result in low quality prediction. In addition to the quality of the dataset, the size of it also matters. Machine learning algorithms are often dependent on having a lot of datapoints to properly differentiate the data that is given, as time used to train and accuracy is often closely tied [36]. The performance of a classifier depends on the number of datapoints used for training, number of features, and classifier complexity [37], among other properties such as the features chosen and the classification task. The features chosen in the Iris flower example in Section 2.5.2 made it easy to distinguish the classes, with clear differences between them. This is not necessarily the case, as other chosen features might result in classes that seem very similar, making them difficult to distinguish.

The amount of data needed can be said to vary depending on the task, number of classes and complexity of the task. Datapoints are represented by one or several features, where a high complexity task can result in a larger number of features, increasing the dimensionality of the feature vector. The amount of datapoints that is needed to properly distinguish the different classes increases exponentially with the dimensionality of the feature vectors [37]. This is called the *curse of dimensionality*. Therefore, one can say that there is a correlation between number of datapoints needed and feature vector dimensionality. A rule of thumb is to use at least five to ten times as many training datapoints per class, as the dimensionality of the feature vector [38]. An increase of feature vector dimensionality, as a result of task complexity, can be seen in a comparison of the feature vectors from the classification examples for Iris flowers and positive and negative numbers, in Sections 2.5.2 and 2.5.1.

2.6 Previous work on EEG and EOG eye movement classification

This section presents solutions for EEG and EOG eye movement classification provided by other papers. Some of the solutions are designed for energy efficiency where self-made algorithms are used to classify the movements, while others use machine learning algorithms as classifiers. Some of the terminology and details presented in this section will be explained later in Chapter 4.

[39] **Ker-Jiun Wang, Lan Zhang, Bo Luan, Hsiao-Wei Tung, Quanfeng Liu, Jiacheng Wei, Mingui Sun and Zhi-Hong Mao (2017). Brain-computer interface combining eye saccade two-electrode EEG signals and voice cues to improve the maneuverability of wheelchair.** A two-electrode EEG system combining eye movement classification and a voice-menu. Eye movements are used to access a voice menu giving voice cues, which is proposed used in a system to control several things such as wheelchairs, TV, smart lights and smart doors. There are in total four classes: Looking straight, looking to the right, looking to the left and blinking. Feature extraction is done with a method called Independent Component Analysis (ICA), while classification is performed with two machine learning algorithms: Support Vector Machine (SVM) and K-Nearest Neighbors (KNN). The KNN algorithm shows slightly better results, and has an average accuracy of around 97%, while SVM shows an average accuracy of around 96%. The Data-processing is done with a bandpass filter from $1-45$ Hz. Thirteen participants (3 females, 10 males) with a mean age of 28 are present in the experimental procedure. The procedure uses a red ball on a screen that is controlled with a joystick and is used to label the eye movements. Each participant executes two tasks, one while looking in horizontal directions and one task for blinking. Horizontal movements are performed around every second, while blinking is performed every two seconds. Each task lasts for 90 seconds, giving a dataset with around 45 datapoints for each class with a total of around 135 datapoints.

[40] **Chi-Hsuan Hsieh and Yuan-Hao Huang (2015). Low-Complexity EEG-Based Eye Movement Classification Using Extended Moving Difference Filter and Pulse Width Demodulation.** This paper presents a low-complexity eye-classification scheme using a self-made algorithm instead of machine learning. The directions classified are right-glancing, left-glancing, up-glancing and down-glancing. The electrode placements used are F7 and F8 for right and left glances, while AF3 and AF4 are used for up and down glances. A low complexity Extended Moving Difference filter is used as edge detection, with Pulse Width Modulation (PWM) and demodulation used to differentiate between the movements and blinks. Positive and negative threshold values for an edge detector is used to detect events. An algorithm uses the timing of events together with the PWM signal to classify the direction corresponding to the event. Blinks are considered parasitic and are removed by setting a threshold where an event is discarded if the duration is less than 0.3s. The presented method in this paper only uses O(N) additions

where N is the processing length of the EEG signal, making it a low complexity solution with an average classification accuracy of 89%.

[41] **Sherif M. Abdelfattah, Kathryn E. Merrick and Hussein A. Abbass (2017). Eye movements as information markers in EEG data.** This paper presents a way to classify the eye-movements up, left, right and down by using 19 channels. Normalization is performed to remove the DC offset present in the electrodes, and Fast Fourier Transform (FFT) is used on the normalized signal to exclude frequencies outside the $0-42$ Hz range. A total of 78 features are presented based on four different properties. A Decision Tree classifier is used to find the best partition of the 78 features and shows that the Theta-Beta ratio (TBR) were the best performing features. A total of 19 TBR features are then used to train three different classifiers: Multilayer Perceptron with 10 hidden layers, Logistic Regression and Random Forest. A voting mechanism is introduced to further improve the accuracy, by looking at the dominant label from the different classifier predictions. Introducing the voting mechanism increases the accuracy from 86.2% to 90.1%. The dataset was created with ten subjects (5 females, 5 males) with an average age of 25. A GUI was created to make the instructions clear. The dataset consists of 15 datapoints in each direction, per subject. Resulting in a dataset of 600 datapoints, 150 per direction.

[42] **Hai Thanh Nguyen, Nguyen Trung, Vo Toi and Van-Su Tran (2013). An autoregressive Neural Network for Recognition of Eye Commands in an EEG-Controlled Wheelchair**. This paper presents a system using an Auto Regression (AR) model with a neural network to distinguish looking straight ahead, blinking, looking to the left and right. Data was recorded through 3 channels (Fp1, F7, F8) and an AR model of second order was used where two coefficients were calculated for each channel resulting in a total of 6 features. The output of the neural network was used to control an electric wheelchair, where the online performance was based on observation of wheelchair control. A controller was designed with a sequence of blinks and eye movements. 3 blinks drove the wheelchair forwards, 4 blinks drove backwards, 2 blinks stopped the wheelchair, looking to the left turned left and looking to the right turned right. The online control of the wheelchair showed an average accuracy of 94%.

[43] **Soumya Sen Gupta, Sumit Soman, P Govind Raj, Rishi Prakash, S Sailaja, and Rupam Bor-gohain (2012). Detecting eye movements in EEG for controlling devices.** This paper presents a system using four electrodes (AF3, AF4, F7, F8), a bandpass filter from 0.5 to 3 Hz

using a 5th order Butterworth filter and the Common Spatial Pattern (CSP) algorithm for feature extraction. SVM is used for classification with four classes, staring straight, right, left and blink. They claim an online accuracy of "around 95%".

[44] **Abdelkader Nasreddine Belkacem, Duk Shin, Hiroyuki Kambara, Natsue Yoshimura and Yasuharu Koike (2015). Online classification algorithm for eye-movement-based communication systems using two temporal eeg sensors.** This paper created a GUI solution with a moving ball to instruct the test subjects. Electrode placements are F7 and F8 and the recorded signal is then filtered with a bandpass filter from 0.5 to 100 Hz using an 8th order Butterworth filter and a 4th order notch filter with stop band from 48 to 52 Hz. Feature extraction is performed by wavelet transform and classification is done with a self-designed algorithm with six classes. The classes were up, down, left, right, straight, and blink. They achieved an online average accuracy of 85%.

[45] **Chi-Hsuan Hsieh, Hao-Ping Chu and Yuan-Hao Huang (2014). An hmm-based eye movement detection system using eeg brain-computer interface.** This system uses a Hidden Markov model with the Viterbi algorithm to realize a low complexity eye-movement detector. They use channels F7, F8, AF3 and AF4. Noise is removed with a low pass filter. Feature extraction is performed with an Independent Component Analysis (ICA) algorithm and then the DC offset is removed with an Extended Moving Difference filter. They implemented the Hidden Markov Model on a FPGA with an achieved 88.6% online detection rate without using machine learning. They claim lower classification complexity than [43]. However, this classification complexity does not include pre-processing and ICA decomposition, as this is performed in MATLAB before it is sent to the Field-programmable gate array (FPGA).

[46] **Rafael Barea, Luciano Boquete, Sergio Ortega, Elena López and JM Rodríguez-Ascariz (2012). EOG-based eye movements codification for human computer interaction.** This paper presents a system that uses wavelet transform and a neural network to detect different angles of eye movements and show an error of less than 2° during long periods of use. However, the task was not to classify horizontal and vertical eye movements, but to distinguish between small involuntary eye movements, voluntary eye movements and the angle of the voluntary movement. The system tries to classify the different angles of eye movements to allow for an eye detection

system which could be used in everyday life. Pre-processing is done with a 0.054 to 35 Hz band-pass filter.

2.7 What remains to be done?

As seen in the literature search there are many different methods that can be used to solve the eye movement classification problem. Some showed success using machine learning, while others created solutions without the use of machine learning at all. It seems that a specific method or algorithm is used to extract the features in the papers, which means that all the features used are of the same type (spectral, time series etc.). Some papers solely focused on using machine learning and getting good classifier performance, while others showed systems without machine learning where every component was designed with energy efficiency in mind. What has yet to be seen, is a proposed list of mixed type features used with machine learning, where a selection of these features is based on both energy efficiency and classifier accuracy. The next chapter will introduce how the system in this work is built up by different components.

Chapter 3

System Design

This chapter will start with a brief overview of the system. It will then go in depth into essential parts of the system, excluding controller, features and classification with machine learning. The main purpose of this system is to train a machine learning classifier with recorded EEG data and use the trained classifier on data captured in real-time to control a drone. The system design is on the software side and only utilizes hardware, APIs and software libraries that are open source. The sections are a mix of work done in this thesis, work done in the previous project [5] and work from others. Section 3.1 starts by presenting the overall system with work both done in this thesis and by others. Section 3.2 describes equipment used in this thesis, which is other's work, but examples have been created in Section 3.2.2 to argument for placement of electrodes. Section 3.3 describes how the datasets were created specifically for this thesis.

3.1 System overview

The overall system is divided into four different main components. Therefore, this section is divided into four subsections explaining each of the main components of the overall system. Figure 3.1 shows a high-level block diagram of the complete system, establishing naming conventions for all signals in the system.

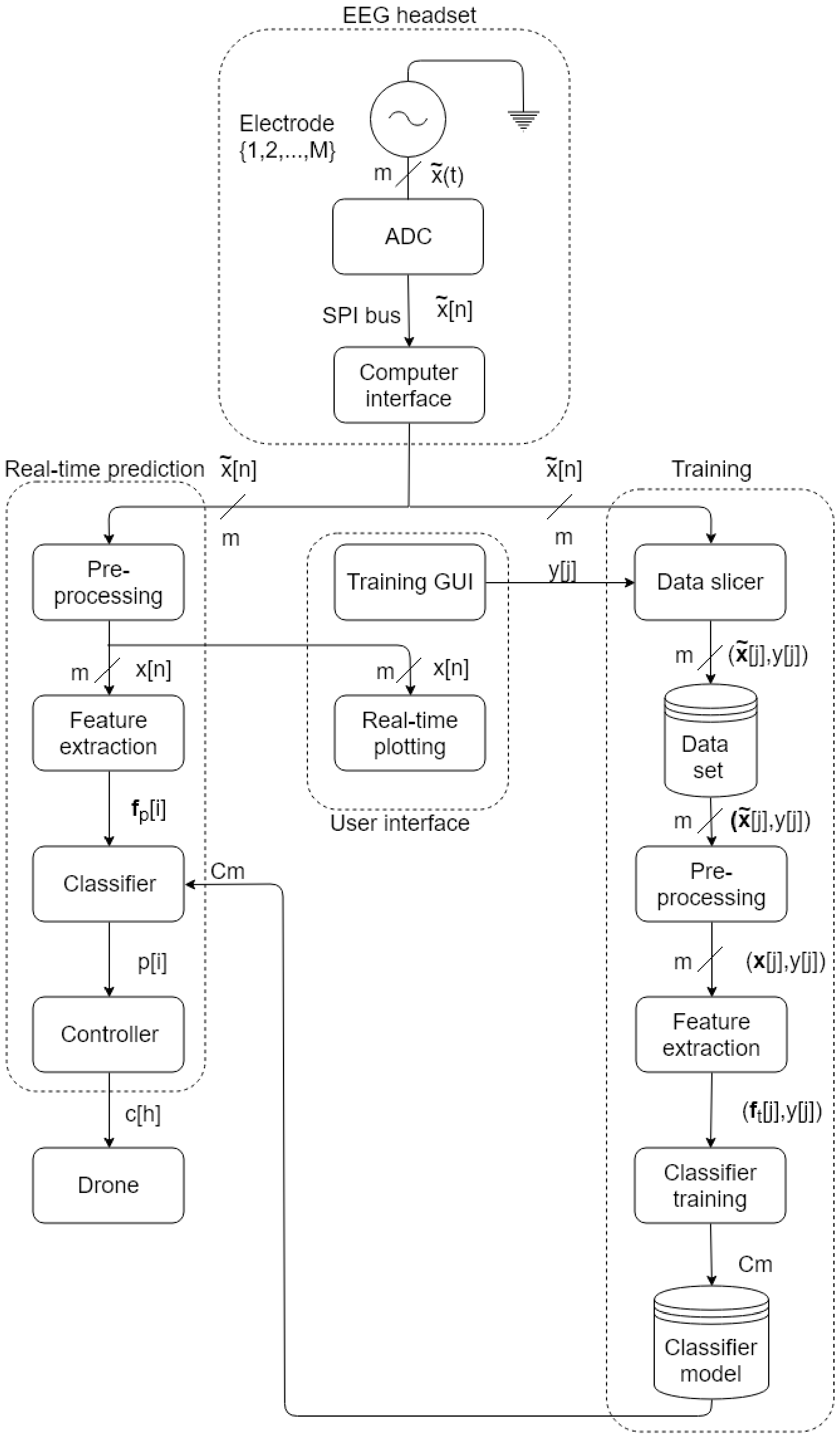


Figure 3.1: Overview of the complete system topology and the signal propagation of the proposed solution.

Figure 3.1 shows a conceptual diagram over the system. Main components are drawn in dashed lines and sub blocks are drawn in solid lines. The "EEG headset" block is a commercially available BCI system and the EEG data is available from an API provided by the manufacturer. The sub block architecture of the "Real-time prediction" and "Training blocks" was designed in the previous project [5] and is used in this thesis with some changes. The "User interface" main block and "Pre-processing" sub block were designed and implemented in the previous project [5] and are used only with minor implementation changes.

The main contribution regarding system design from this work is the remaining sub blocks, with special focus on the "Feature extraction", "Classifier" and "Controller" blocks, as these are regarded as critical to system performance. The "Feature extraction" and "Classifier" blocks are designed around methods and toolboxes already created and will be explained in detail in Chapter 4. The "Dataset", "Data slicer" and "Controller" blocks are created from scratch in this work, and are explained in more detail in Sections 3.3.4, 3.3.3 and 6.2. Others have created the "Drone" block [47], while "Training GUI" and "Real-time plotting" was implemented in the previous project [5].

EEG headset

Starting from the top with the "EEG headset" block, the electrodes are represented by a voltage source, where $m \in \{1, 2, ..., M\}$ and M is the number of electrodes. When referencing a specific channel, subscript notation is used, ex. $\widetilde{x}_m[n]$. $\widetilde{x}(t)$ is an analog signal that is measured by the scalp-electrodes and needs to be converted into a digital format. The result from passing $\widetilde{x}(t)$ through the Analog-to-Digital converter (ADC) is $\widetilde{x}[n]$ which is derived from Equation 3.1.

$$\widetilde{x}[n] = \widetilde{x}(nT_s), \tag{3.1}$$

where:

- T_s = sampling period
- $f_s = 1/T_s$ is the sampling frequency

Real-time prediction

Continuing with the "Real-time prediction" block, which takes $\tilde{\mathbf{x}}[n]$ as an input. The first stage is "Pre-processing" where DC offset and known noise components of the signal are removed. The output of the Pre-processed signal is $x[n]$. The next stage is "Feature extraction" where a sliding window, of length $l_{\mathrm{w,c}}$, of data $x[n]$ is abstracted into a feature vector named $\mathbf{f}_p[i]$ (p for prediction). Please note that the indexing is changed, from n to i, as the rate of the "Feature-extraction" f_f is not necessarily the same as the sampling frequency f_s. If the sliding window length is set to be such that $l_{\mathrm{w,c}} > \frac{f_s}{f_f}$, the windows will overlap each other. The channel index m has also disappeared with the "Feature extraction" block, as the signals from the different channels now are represented as features in a vector $\mathbf{f}_p[i]$ calculated from a selection of the channels.

For each index of i the "Classifier" will perform a classification with input $\mathbf{f}_p[i]$, based on the classifier model Cm. The output from the classifier, $p[i]$, is the class labels predicted for an eye movement. The "Controller" will then in turn interpret these predictions from the classifier to control the drone with the command signal $c[h]$. The reason for the new index h is that the control signal is not produced at the same rate as the classification. The connection between labels and classes will be shown in Section 3.3.1.

Training

Next, we have the "Training" block that also takes $\tilde{\mathbf{x}}[n]$ as an input. The first stage is the "Data slicer", which slices a window of length $l_{\mathrm{w,tot}}$ from $\tilde{\mathbf{x}}[n]$ and saves it on disk, this is done each time it receives a label $y[j]$ from the "Training GUI". Each sliced window is a datapoint in the complete dataset of eye movements. The sliced window (which is saved on disk) is called $\tilde{\mathbf{x}}[j]$, where $y[j] \in \{0,2,4,5,6,8\}$ is the class label for that sliced window and j is dataset index. Storing raw data $\tilde{\mathbf{x}}[n]$ instead of pre-processed data $\mathbf{x}[n]$ makes it possible to change the pre-processing in the future without having to make a new dataset.

Then, we have "Pre-processing", which is nearly identical to the "Pre-processing" stage in the "real-time prediction" block, except that it also slices off some part of the start and the end of the window. It was found in this work to be necessary to remove the initialization period of the digital filter from the data, and for some minor alignment adjustments during training. The removal of initialization periods for the digital filter makes the data look like it is taken from a continuous stream of pre-processed data, making it similar to what it will experience in a real-time prediction setting.

The "Feature extraction" in the training block is identical to the "Feature extraction" in the real-time block and outputs a feature vector named $\mathbf{f}_t[j]$ (t for training). The last stage is the "Classifier training" that uses all the labeled feature vectors, $(\mathbf{f}_t[j], y[j])$, to train a classifier model Cm and save it to disk. More about feature extraction and classifier training will be presented in Chapter 4.

User interface

The "Training GUI" will be introduced later in this chapter. The real-time plotting continuously plots the 2000 last arriving pre-processed samples. This corresponds to a window of 8 seconds.

3.2 Headset

The headset used is made by OpenBCI and is the Ultracortex IV with the OpenBCI Cyton board [48, 49]; a low-cost, programmable, open source BCI-platform. It is built around the Texas Instruments ADS 1299 IC, which is a low noise 24-bit ADC with 8 channels specifically designed for bio-potential measurements [50]. Communication between board and computer happens through Bluetooth, making it possible to not have any physical connection to the main grid. A block diagram showing its interaction can be seen in Figure 3.2 and the hardware can be seen in Figure 3.3.

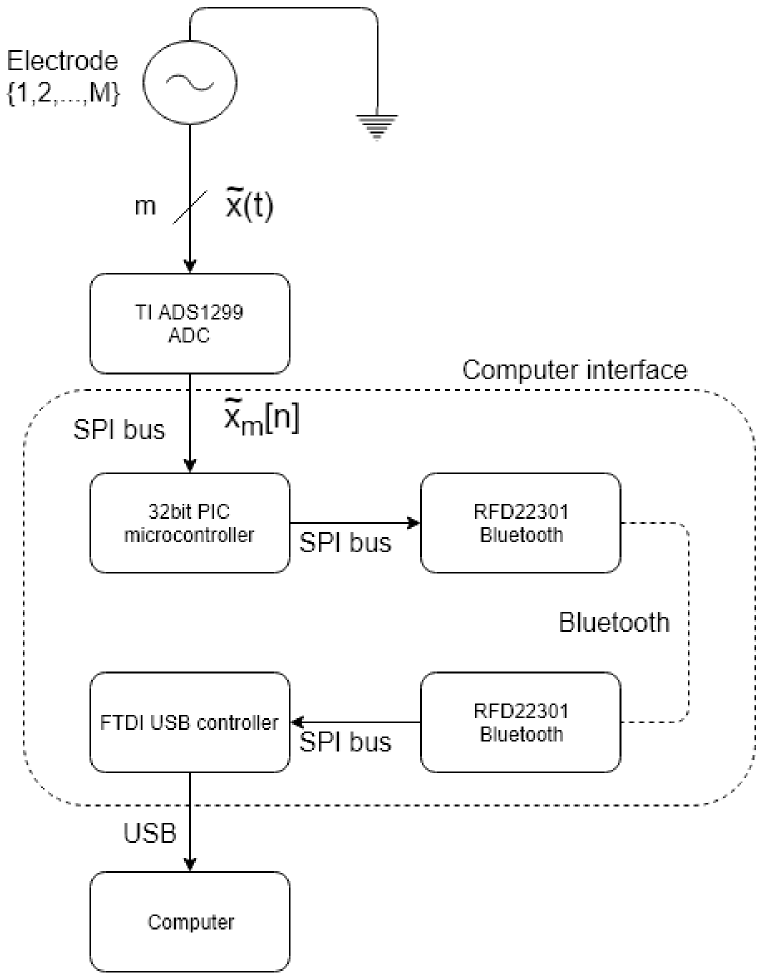


Figure 3.2: Block diagram of the hardware and how the signals are propagated. Hardware related to the Cyton board is in the top row and hardware related to the USB dongle in the bottom row.

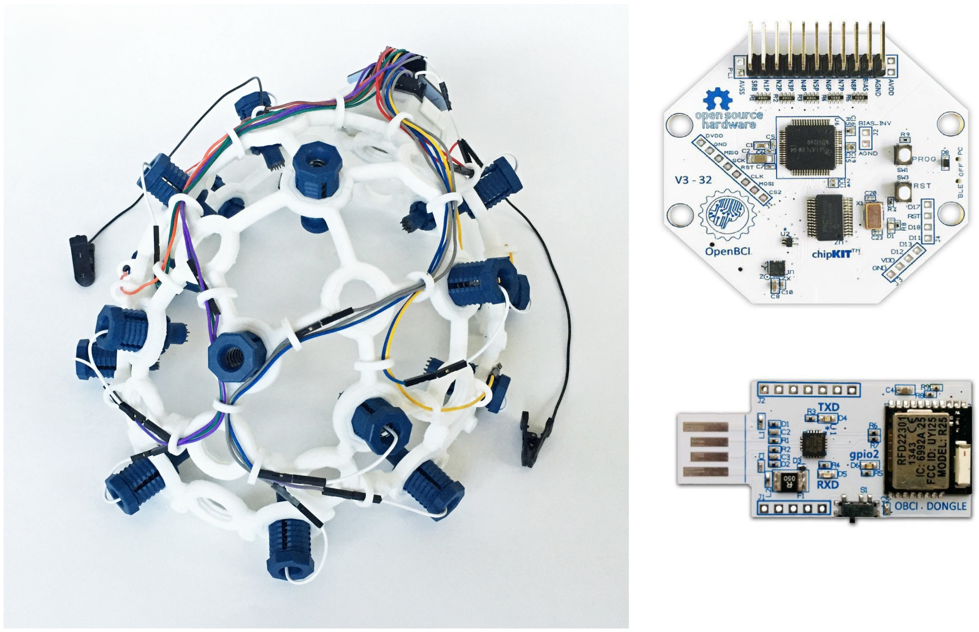


Figure 3.3: Shows the equipment used to measure the EEG data. Headset with attached electrodes to the left [48], Cyton board in the upper right corner and the USB dongle for Bluetooth communication in the lower right corner [51].

Data sampled by the ADC goes unfiltered, with a sample-rate of $f_s = 250$ Hz, through the onboard PIC microcontroller and is transmitted by a radio to the USB dongle. From there, the data is available in Python via an Application Programming Interface (API) from OpenBCI [52].

3.2.1 Electrodes

The electrodes used in this kit are dry Ag/AgCl electrodes and can be seen in Figure 3.4 [53]. Dry electrodes are very convenient as there is no need for any gel or other liquid to get good signals from the skin.

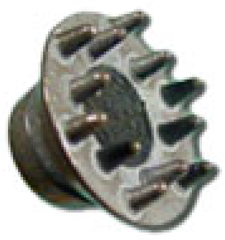


Figure 3.4: Picture of a dry electrode from Florida Research [53]. There are in total eight of these attached to the headset.

Evaluating the signal quality of the electrodes is complicated. To give some information about this, one thing we can do is to look at an experiment created to show the frequency components in $\widetilde{x}_1[n]$. Figure 3.5 shows the spectrum plot while a subject looks straight ahead, without filtering of data.

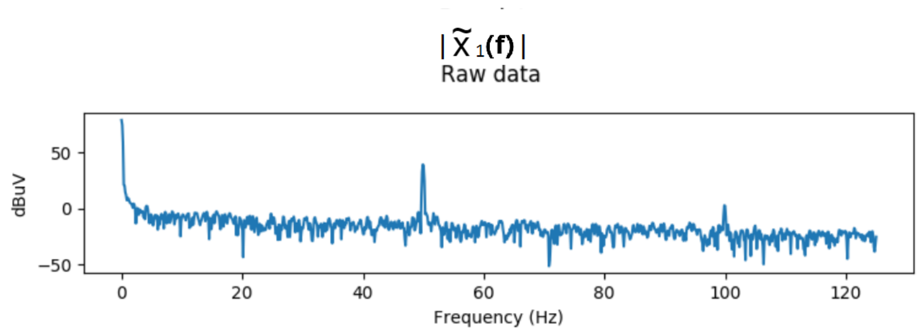


Figure 3.5: Amplitude response $|\widetilde{X}_1(f)|$ with y-axis in dBμV. The recording is made when holding the eyes completely still.

The plot in Figure 3.5 is a Discrete Fourier Transform (DFT) plot. The signal is multiplied with a Blackman window [54], defined in Equation 3.2, before the transformation to reduce spectral leakage [55].

$$w[n] = 0.42 - 0.5\cos\left(\frac{2\pi n}{L}\right) + 0.08\cos\left(\frac{4\pi n}{L}\right) \tag{3.2}$$

The transform is defined by Equation 3.3 [56]:

$$\widetilde{X}(f) = \sum_{n=0}^{L-1} \widetilde{x}[n]e^{-i\frac{2\pi}{L}\frac{f}{fs}} w[n] \tag{3.3}$$

The frequency band is restricted to only show $0-125$ Hz components as 125 Hz is the highest representable frequency with a sampling rate of $f_s = 250$ Hz. This range of $0-125$ Hz covers the whole known brain wave frequency band as shown in Table 2.1.

We can see that the signal has some significant 0, 50 and 100 Hz components that were found to be noise in the previous project [5].

3.2.2 Electrode placements

There are several options when it comes to placement of the electrodes. Figure 3.6 shows the electrode positions supported by the 10/20 electrode placement system. Signal electrodes used in this work are marked in yellow and grounding electrodes are marked in green.

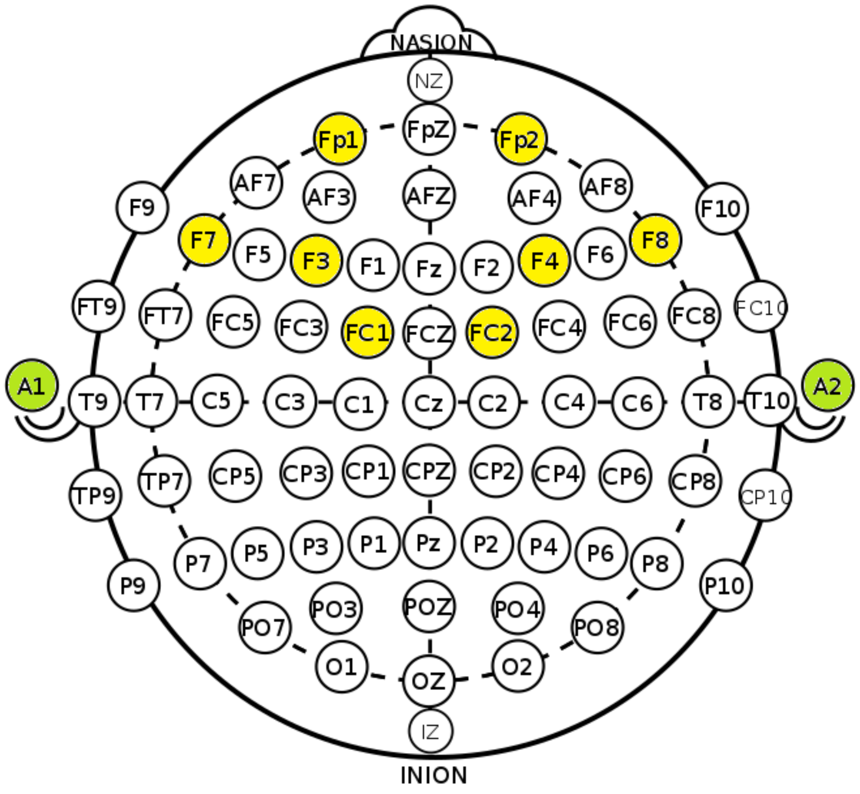


Figure 3.6: Illustration of the 10/20 electrode placement system with utilized positions marked in yellow and green [57].

The $Fp1$, $Fp2$, $F7$ and $F8$ placements are chosen because they are good at detecting ocular artifacts, as they are close to the eyes. $F3$, $F4$, $FC1$ and $FC2$ are electrode placements related to the Frontal Eye Field (FEF) area. The FEF region is responsible for both small involuntary and larger voluntary eye movements [14], as mentioned in Section 2.2.

To illustrate that the $Fp1$, $Fp2$, $F7$ and $F8$ positions are good for detecting ocular artifacts, a model of the eye and electrodes was made in this work. As mentioned in Section 2.3 the eye can be modelled as a dipole [17], with positive charge at the cornea and negative charge at the retina. When we rotate the ocular dipole, the change in charge is best detected by an electrode close to the rotational plane of the dipole and far from the axis of rotation, given that the rotational axis is orthogonal to the dipole axis. This is illustrated in Figure 3.7, showing two approximated electrode placements ($Fp1$ and $F7$). Using this as a basis, calculations can be made to find the usefulness of the electrode placements for the different eye movements. These calculations are done as part of this work.

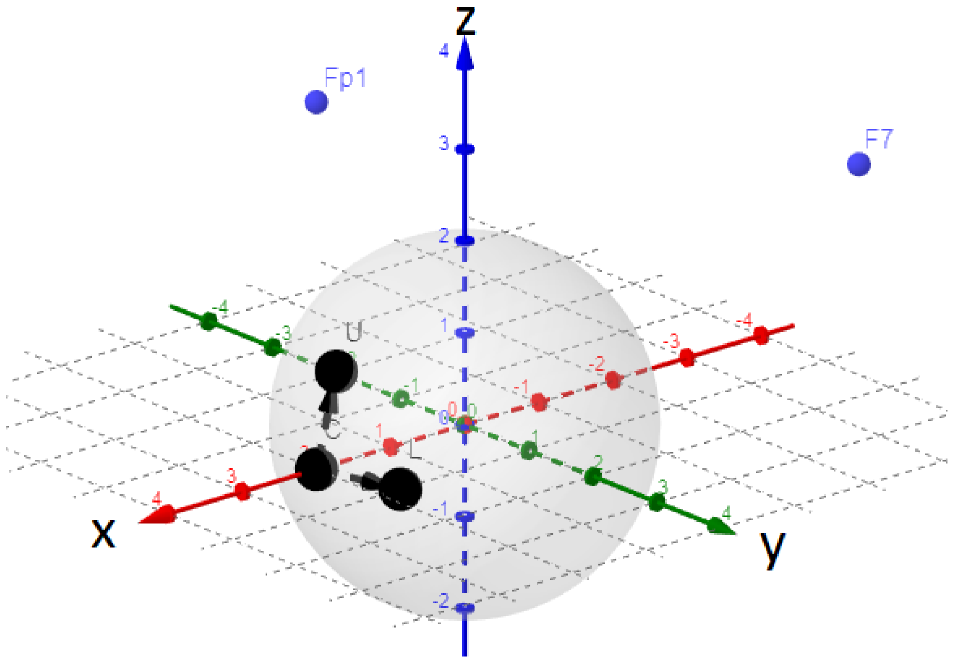


Figure 3.7: Illustration of 30° up and left movements of the left eye, relative to approximate electrode positions.

The eye can be modeled as a sphere with $r = 2$ and center in the origin, with the cornea placed in resting position at $C = (2,0,0)$. If the electrode placements are roughly approximated, the coordinates for the electrodes can be given as $Fp1 = (2,0,4)$ and $F7 = (-1,5,4)$. It is now possible to calculate the change in distance from the Cornea to the electrodes with a 30° movement towards left, called L and a 30° upward movement called U. The created model presented in Figure 3.7 is generalized, meaning that it can be used for all electrode placements in the 10/20 electrode placement system.

To calculate the positions for U and L a rotation matrix is utilized. For the Up position the rotation is performed around the y-axis, where the calculation for the rotation can be seen in Equation 3.4.

$$U = \begin{bmatrix} \cos -30^\circ & 0 & \sin -30^\circ \\ 0 & 1 & 0 \\ -\sin -30^\circ & 0 & \cos -30^\circ \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \sqrt{3} \\ 0 \\ 1 \end{bmatrix} \tag{3.4}$$

For the Left position the rotation is performed around the z-axis, where the calculation for the rotation can be seen in Equation 3.5

$$L = \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ & 0 \\ -\sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \sqrt{3} \\ 1 \\ 0 \end{bmatrix} \tag{3.5}$$

Next, the distances between the electrode placements and the Center, Up and Right positions are needed. This can be calculated with the Pythagorean theorem and gives the following results in Equation 3.6.

$$\begin{gathered} |CFp1| = 4 \\ |CF7| = 7.07 \\ |UFp1| = 3.01 \\ |UF7| = 6.44 \\ |LFp1| = 4.13 \\ |LF7| = 6.28 \end{gathered} \tag{3.6}$$

The coordinates of the electrodes and the Center, Up and Right positions are:

- $Fp1 = (2, 0, 4)$
- $F7 = (-1, 5, 4)$
- $C = (2, 0, 0)$
- $U = (\sqrt{3}, 0, 1)$
- $L = (\sqrt{3}, 1, 0)$

The change in distance to the respective electrodes can be seen in Equation 3.7

$$\begin{aligned}\Delta Fp1U &= 0.99\\ \Delta F7U &= 0.63\\ \Delta Fp1L &= -0.13\\ \Delta F7L &= 0.79 \end{aligned} \tag{3.7}$$

When interpreting these results, it is important to keep in mind that the electrode placements in the model are rough estimates. The distances from $C = (2,0,0)$ to the electrode placements in the model probably differs from the real-world system. In reality, having one electrode much closer to the eye will result in the strongest changes in the closest electrode. The model presented in this work was made to illustrate that placing electrodes in different directions relative to the eye, can make it more suitable for horizontal and vertical eye movements. The distance from the electrodes to the eye should only affect the magnitude of the changes for each electrode, not which movement shows the absolute greatest change in the electrode.

This means that it is not necessarily a good idea to compare the electrode positions with each other, but rather how the electrode potential changes with horizontal and vertical eye movements. When we compare the electrode placement $Fp1$ for the U and L movements, we can see that it has the greatest change in distance for the U movement. When comparing the electrode placement $F7$ for the U and L movement, we can see that it has the greatest change in distance for the L movement. This difference is valuable for a classification situation and it can therefore be concluded that $Fp1$ and $Fp2$ are good for measuring vertical eye movements, while $F7$ and $F8$ are good for measuring horizontal eye movements.

As mentioned early in Section 3.2.2, $F3$, $F4$, $FC1$ and $FC2$ are electrode placements of interest related to the Frontal Eye Field (FEF) area and can contain brain signals regarding the eye movements. The FEF region is responsible for both small involuntary and larger voluntary eye movements [14].

Table 3.1 shows the mapping of electrode placements to channels, and their signal names.

Table 3.1: Channels, electrode placements and signals

Channel	Electrode placement	Signal
1	$Fp1$	$\widetilde{x}_1(t)$
2	$Fp2$	$\widetilde{x}_2(t)$
3	$F7$	$\widetilde{x}_3(t)$
4	$F8$	$\widetilde{x}_4(t)$
5	$F3$	$\widetilde{x}_5(t)$
6	$F4$	$\widetilde{x}_6(t)$
7	$Fc1$	$\widetilde{x}_7(t)$
8	$Fc2$	$\widetilde{x}_8(t)$

3.2.3 Pre-processing

The system for pre-processing the data was designed in the previous project [5]. A short summary of the pre-processing can be found in Appendix A. The result was a 14$^{\text{th}}$ order IIR notch filter, with notch frequencies at 0 Hz, 50 Hz and 100 Hz.

3.3 Creating the datasets

An easy and effective method for creating datasets was desired. It was thought to be most effective if a computer instructed the subjects to move the eyes in different directions. By providing instructions through a stimulus and record the EEG signals a period after the instruction. This will be explained in Section 3.3.1. Both visual and audio stimuli are easy to implement, but as the auditory cortex is located much closer to the area of interest than the visual cortex [13], visual stimuli is chosen as this might interfere less with the measurements. This approach was also used by others [39], creating a visual stimulus by using a moving dot on a screen. This approach will also be applied in this work.

Two subjects participated in the experiments for this thesis, both for the creation of datasets and testing of classifiers. Both subjects are right handed males with an average age of 24.5 ± 1.5 years. Both subjects have corrected-to-normal vision and no reported current or past neurological or psychiatric illness.

3.3.1 Graphical User Interface

To get a consistent basis for generating training data, a Graphical User Interface (GUI) is used to control the timing and direction of eye movements performed by the test subjects. This also makes it possible to make a system for automatic labeling and splitting of recorded data into time segments. The visual design of the GUI was made in the previous project [5], but the back-end is from this work. The visual design is shown in Figure 3.8.

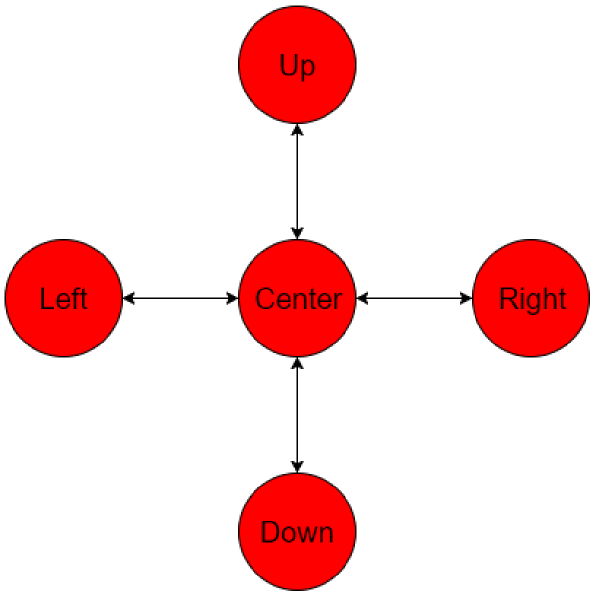


Figure 3.8: Showing how the dot in the GUI can move in different directions.

The idea is that the test subject follows a red dot with the eyes and blinks when the dot disappears. The dot is moving in clockwise sequence of vertical and horizontal directions, returning to center in between movements. The clockwise sequence is chosen over a random sequence to allow the subject to be aware of the direction to look in. This is more similar to the actual use of the system, where the subject always will be aware of the direction to look in when controlling the drone. The dot is held in the uttermost position of a given direction for one second before returning to center. When it has reached center it stays there, still, for two seconds. After that it disappears for half a second and reappears standing still in center for three seconds, before commencing a new movement. The timing of each movement and the number of blinks is found according to experimental findings on what feels comfortable and to minimize the number of *parasitic* blinks. *Parasitic* blinks are considered as blinks that occur without the GUI explicitly telling the test subject to blink. An overview of the class and the associated eye movements can be found in Table 3.2. Names are given to each class and are used throughout the rest of the thesis.

Table 3.2: Classes and their labels

Class	Label
Blink	0
Up	1
Down return	2
Right	3
Left return	4
Straight	5
Right return	6
Left	7
Up return	8
Down	9

With the current speed of the GUI, a dataset with one datapoint for each of the classes can be made every 30 seconds. This corresponds to at least 120 datapoints of each class per hour in theory. The occurrence of the Straight and Blink classes will be four times as large as the rest of the classes, due to the nature of the movements. To achieve this throughput in practice, full concentration must be maintained and no errors can occur (not likely). The throughput of datapoints per hour is therefore expected to decrease in a practical setting as it is hard to maintain the concentration over a longer period. Moving the eyes to the extents of each direction is also tiresome to the subject. The datasets will therefore be created in smaller intervals, where the subject follows instructions from the GUI as long as comfortable.

3.3.2 Setup

The test subjects are positioned in front of a screen in such a manner that the dot is at eye height. The distance to the screen was approximately 15 cm, to ensure that the dot moves out of sight in the different directions (screen size highly affects the distance needed). This ensures maximum peak value of the signal and will make it easier to differentiate between small involuntary movements happening all the time and voluntary movements for control. The reason for larger peaks when moving the eye further away from center is because the change in position is larger, relative to the electrodes. This was mentioned in Section 3.2.2.

3.3.3 Epoching the data

The system for splitting the data into equal sized epochs (time periods) and labelling them is controlled by the "Training GUI" and performed by the "Data Slicer". This is illustrated with the block diagram in Figure 3.9.

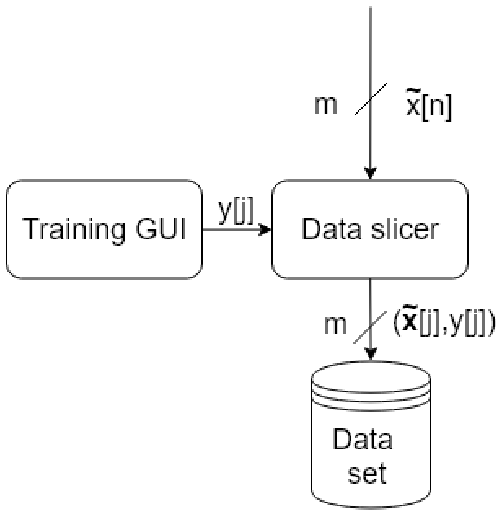


Figure 3.9: Block diagram of the data slicing and labeling operations

It can be seen from Figure 3.9 that the "Training GUI" only sends the label $y[j]$ to the "Data slicer". Each time the GUI has moved the red dot, it spawns a "Data slicer" thread that looks for data with the same time stamp $\tilde{x}t[n]$ as when the thread was started $yt[j]$. It then appends unfiltered data $\tilde{x}[n]$ of length $l_{\text{w,tot}}$ to the dataset file. The epoching of data can be illustrated with Figure 3.10

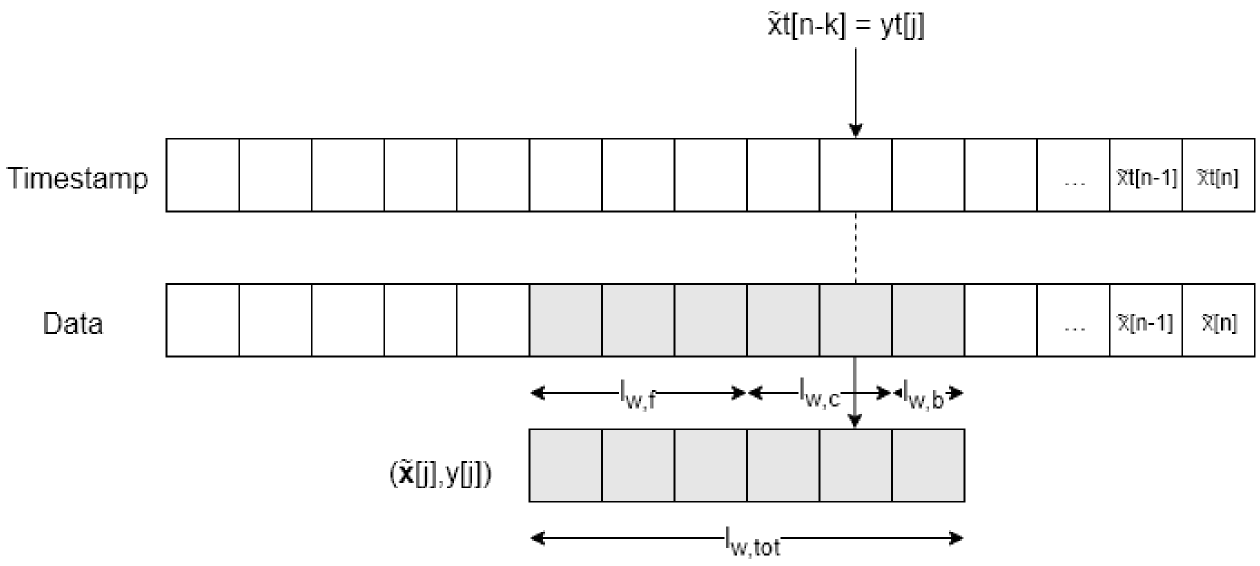


Figure 3.10: An illustration of how the data slicing operation is performed.

The $\tilde{x}[n]$ signal is divided into three sub-windows of different lengths, $l_{\text{w,tot}} = l_{\text{w,f}} + l_{\text{w,c}} + l_{\text{w,b}}$. The sub-windows are called front padding, interesting area and back padding, respectively. The lengths are $l_{\text{w,f}} = 750$ samples, $l_{\text{w,c}} = 250$ samples and $l_{\text{w,b}} = 250$ samples. As a reminder, the sampling frequency is $f_s = 250$ Hz.

The sample with matching time stamp $\tilde{x}t[n] = yt[j]$ is the last sample of the interesting area. The sliced window $\tilde{\mathbf{x}}[j]$ is saved together with the label $y[j]$, which represents the recorded eye movement. The datasets are stored as raw data from $\tilde{x}[n]$, making it possible to change the pre-processing in the future, without having to make a new dataset. The padding allows time for filter initialization and minor alignment adjustments. The padding is removed after pre-processing, and before the data is being used to train the system.

An example of a signal with padding from channel $x_1[t]$, for Down and Down return, can be seen in Figure 3.11.

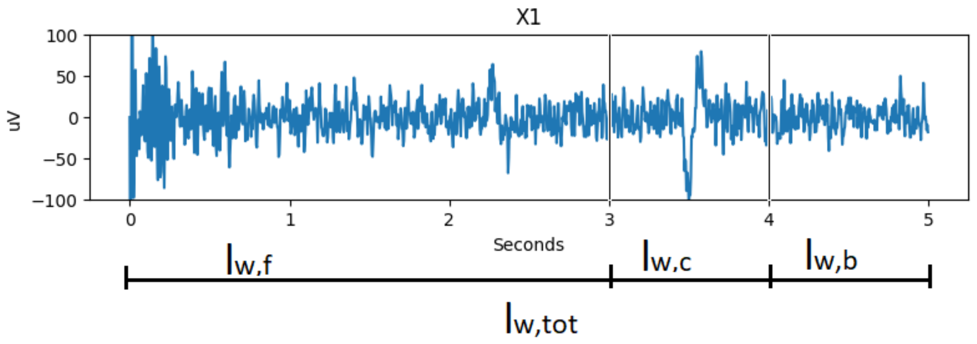


Figure 3.11: A whole window from channel $x_1[t]$ with padding, for the Down and Down return movements.

Figure 3.11 shows the whole window with pre-processing. The same window is stored in the dataset, but as the raw version of the signal. The pre-processing was used for illustration purposes as the raw signal mostly looks like 50 Hz noise, and to show the filter initialization region. One can be see that the filter initialization lasts for ~ 2 seconds, and that it is possible to shift the window $l_{\mathrm{w,c}}$ up to 1 second in each direction.

3.3.4 Dataset

In the dataset, moving the eyes in a direction and then returning back to center is stored as separate movements. This gives a total of 10 classes in the datasets as introduced in Table 3.2.

It was later found that some of these classes and labels, such as Up (1) and Down return (2), could be merged to the same class and label. This is because the relative direction and distance covered when moving the eyes from center and up, and from down to center should be the same. The only thing that differs is the starting point for the movements. Looking up can therefore be considered to be the same as returning to center after a down movement has been performed. This accounts for all the horizontal and vertical directions. The dataset is still stored with the 10 original classes, to maintain the possibilities to use these separate movements in some other classification task. Instead, when the dataset is loaded for training and testing of the classifier, the classes and labels are merged before it is sent to the classifier.

Merging these gives a total of 6 classes and labels. Table 3.3 shows the mapping from the original 10 classes and labels to the 6 new classes and labels.

Table 3.3: Classes and labels before and after merging

Original class	Original label	New class	New label
Blink	0	Blink	0
Up	1	Up	8
Down return	2	Up	8
Right	3	Right	6
Left return	4	Right	6
Straight	5	Straight	5
Right return	6	Left	4
Left	7	Left	4
Up return	8	Down	2
Down	9	Down	2

The classes and labels presented in Table 3.3 are used throughout this thesis. An example of a plot of the Blink class can be seen in Figure 3.12. Plots for the rest of the classes can be found in Appendix B.

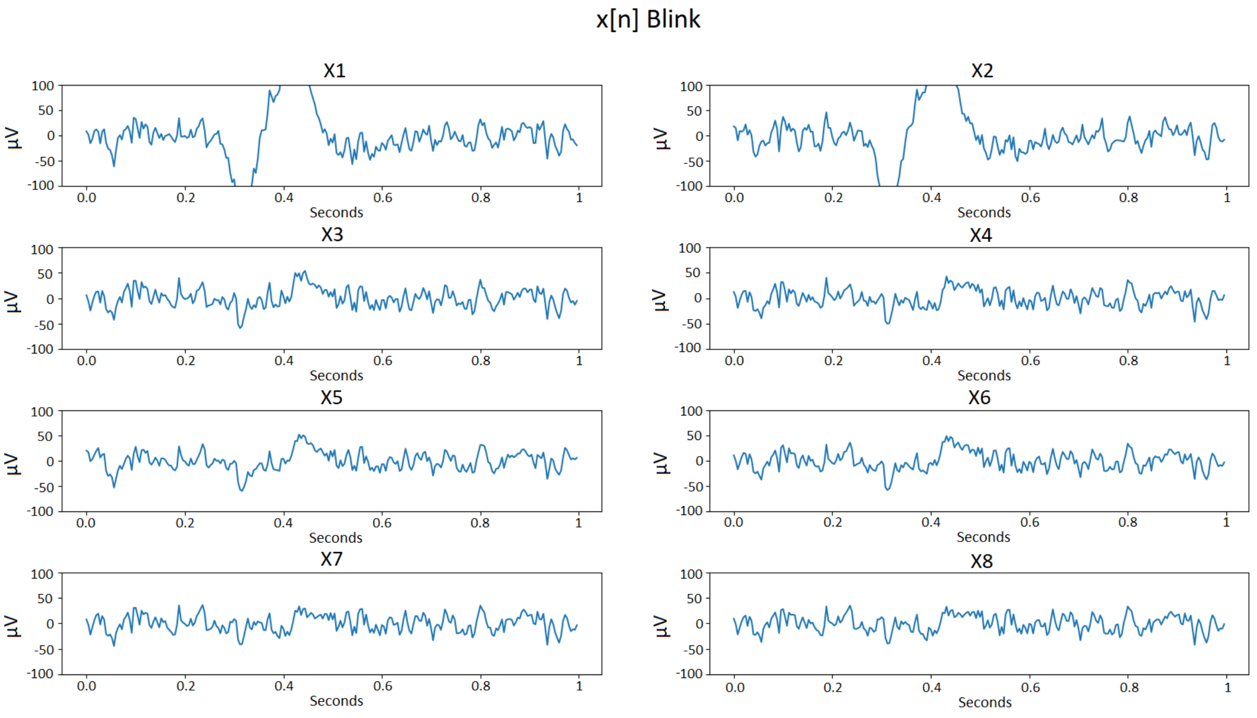


Figure 3.12: Showing a datapoint for the Blink class.

3.3.5 Dataset validation and size

It was considered important to make sure that the data representing the different movements in the datasets were free of errors. To make sure that the dataset contains the desired information, and not something else entirely, all datapoints were plotted as in Figure 3.12 and inspected. Datapoints with obvious variations which seemed to come from sources such as poor electrode connection or movements other than eye movements (facial, arms, legs etc.), were removed from the datasets. As a result of this, the classifier accuracy is likely to fall if such variations are present in the recorded signals during testing. This might require the subjects to stay completely still without performing movements other than eye movements to get high classifier accuracy.

The removal of datapoints resulted in an unbalanced dataset where some classes had a higher number of datapoints than others. To solve this, the number of used datapoints for each class were decided when the dataset was loaded, before training and testing. An uneven balance could make the classifier biased towards one or a few of the classes. Balancing is not always necessary, for example in datasets where the real-world application of the model is to be used on unbalanced data. The problem in this work on the other hand does not favor any eye movements (classes), so having a balanced dataset makes for "fair" classification.

The number of datapoints used when training differs according to the type of tests performed, the number of used datapoints will therefore be introduced in Chapter 5 and Section 6.5. However, the total number of datapoints for the original classes from Table 3.2 contained in the datasets is shown in Table 3.4.

Table 3.4: Number of datapoints in the two subjects' datasets before merging.

Class	Subject 1	Subject 2
Blink	638	805
Up	153	202
Down return	162	314
Right	150	134
Left return	156	141
Straight	625	674
Right return	143	169
Left	156	142
Up return	130	198
Down	134	201
Total	2447	2980

When merging the classes, only the lowest count of the two classes merged is counted. This is because it is desired to have an equal amount of datapoints from looking in a direction and its equal from the opposite return (Up and Down return for example). Table 3.5 shows the number of datapoints of the resulting classes after merging.

Table 3.5: Number of datapoints in the two subjects' datasets after merging.

Class	Subject 1	Subject 2
Blink	638	805
Down	260	396
Left	286	284
Straight	625	674
Right	300	268
Up	306	404
Total	2415	2831

Each dataset was inspected by one inspector each, meaning that the evaluation of the datapoints is somewhat subjective. This can affect the results somewhat during testing shown in Chapter 5 and Section 6.5. When the two datasets are used, it is made sure that the number of datapoints chosen from each dataset is the same. So even if Subject 2 has a bigger dataset, the classifiers will be trained on the same number of datapoints from both subjects. Using the lowest occurrence of a class from the subjects, the number of datapoints for each movement is 260, giving a total of 1560. The reason for using the same amount of datapoints for both subjects is so that the classifiers will be trained and tested on the same amount of data. The results will therefore present the differences in the datapoints, and not the amount of training the classifier has done.

3.3.6 Reducing window size

In the example shown in Section 3.3.3, the window size was set to $l_{\mathrm{w,c}} = 250$ samples. It is desirable to be able to vary the window size in order to focus on different parts of the signal. Reducing the window size could make sure that no other data than the data representing an event is contained in the window and allow for movements to be performed in quick succession.

When reducing the window length, it is necessary to make sure that the samples representing the movement is located within the new reduced window. For reductions down to 200 samples, this is usually not a problem as it equals a 25-sample reduction in each end of the window, as the original alignment of the data places the movement well within the 200 central samples of the window. Any further reduction might lead to parts of the movement being left out of the window. To avoid this, one can use a method for dynamically center the samples representing the movement.

The dynamic centering can be performed by smoothing the data with a Savitzky-Golay filter [58, 59] and then finding the index with maximum absolute value, called peak-index p_i. After the peak-index has been found, this can be used as the center of the new reduced window. The Savitzky-Golay filter is chosen for properties such as a low time shift and good preservation of the original signal shape. An example of finding the peak-index p_i with a problematic window from the dataset can be seen in Figure 3.13.

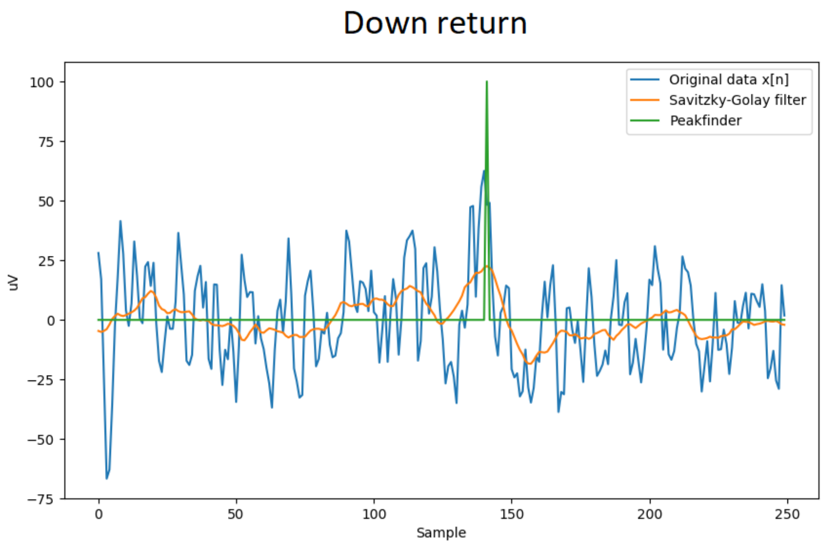


Figure 3.13: Pre-processed data x[n], smoothed pre-processed data and peak finder

Figure 3.13 illustrates why the smoothing is necessary. The largest absolute value is located around index 0, which would have become the center of the window without use of the smoothing filter. This is however not the center of the movement, which is located around index 145. This is known to be the center of the movement as it has been observed that the Down return class is characterized by a high positive peak-shape. It is also supported by literature that eye movements create a change in potential [17], which are peak-shaped when high-pass filtered. The samples around this peak-shape are the desired data as the eye movement is of focus in this thesis, not the brainwaves triggering the movement. The smoothing filter makes it possible to find the peak-index representing the center of the specific eye movement and ignore the largest absolute value at around index 0. The parameters for the smoothing filter are: polynomial order = 2 and window = 35. After the peak-index p_i has been found it only remains to adjust the window sizes accordingly, shown in Equation 3.8. The p_i index is referenced to the whole window $l_{w,tot}$.

$$l_{w,f} = p_i - \frac{l_{w,c}}{2}$$

$$l_{w,b} = l_{w,tot} - \left(p_i + \frac{l_{w,c}}{2}\right) \tag{3.8}$$

If one does not wish to center the samples representing the eye movements dynamically, but just want to statically reduce the window size $l_{\mathrm{w,c}}$, it is possible to just set $p_i = 875$, as this corresponds to the center of $l_{\mathrm{w,c}}$ before any window size reduction. The window size $l_{w,c}$ can now be reduced down to any size, with or without centering, as can be seen in later chapters. The most common setting for work in this thesis was no centering with window size $l_{\mathrm{w,c}} = 250 \vee 200$, but settings with centering and window size down to $l_{\mathrm{w,c}} = 50$ are also present. Figure 3.14 shows a block diagram for the dataset pre-processing.

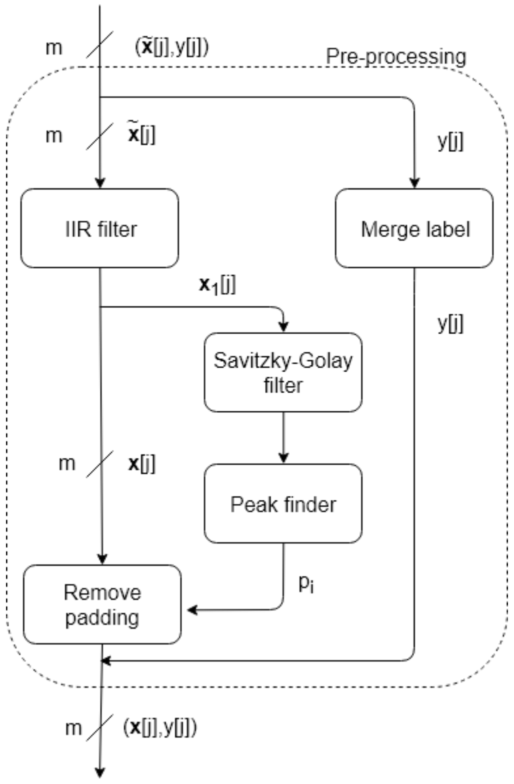


Figure 3.14: Block diagram for dataset pre-processing

It can be seen from Figure 3.14 that only data from channel x_1 is sent into the smoothing filter and that the peak index p_i found in channel x_1 is used to center all the channels equally. This is because x_1 has been observed to have the largest average absolute values, across all the different eye movements. The peak-index p_i could, however, also have been from x_2 or the average of x_1 and x_2. It is also worth mentioning that the "Merge label" block in Figure 3.14 is responsible for the merging of classes and labels when loading the datasets, as mentioned in Section 3.3.4.

3.4 System design summary

This chapter introduced the different components of the system, a method for gathering data, splitting the data into epochs (time periods) and adjusting the epochs. A GUI was created to instruct subjects on the eye movements they should perform, allowing the data to be created with a theoretical speed of 120 datapoints an hour for each class (480 for Straight and Blink). Balancing the classes and the two datasets makes sure that all classes are considered equally important, ensuring a fair comparison of the two subjects. The output of the system in this chapter is data, ready to be represented by features and used with a machine learning algorithm.

Chapter 4

Machine Learning

This chapter introduces machine learning in more detail and shows how it is implemented in this thesis. Terminology and methods mentioned in this chapter builds on concepts introduced earlier in Section 2.5. To freshen up the concepts of machine learning a few terms are repeated:

- dataset, all datapoints and labels
- datapoint, a data element in the dataset, in this thesis each datapoint is an eye movement
- feature vector, the representation of the datapoints where the feature vector consists of one or several features
- labels, identifier of the classes the different datapoints belong to
- prediction, the output decision when testing the classifier

As in Section 2.5, this chapter provides an interpretation of different methods, concepts and expressions from different literature that was utilized in this work. It also presents some important scoring metrics for classifiers that will be used throughout the thesis. The chapter is *not* based on any "new" findings from any work in this thesis. However, figures and examples have been created to better illustrate some of the concepts in hope of making it easier to understand.

The machine learning toolkit used in this work is Scikit-learn [6], which is frequently referred to through the next chapters. Scikit-learn provides high level implementations of many popular machine learning algorithms, data processing methods and feature selection methods among many other functionalities.

4.1 Extracting and selecting features

In this work, it might be necessary to abstract the data by extracting features, to reduce the dimensionality of the feature vector [37]. The features that are used has high impact on the performance of the classifier, measured by scoring metrics introduced later in Section 4.3.1. Finding features to represent the eye movements and to evaluate the quality of these features is important to get the best possible performance of a classifier.

4.1.1 Feature extraction

A crucial and difficult task of extracting features is to represent the original data without too much loss of relevant information. By knowing the problem the classifier tries to solve, it is also possible to extract features that can bring classification improvement [33]. While it is easy to extract features from data, it is not easy to extract features that are usable to distinguish between the desired events. Finding the features that best describes the data for a specific problem can be a demanding task. Usually, the most secure way to extract high quality features is to have an expert in the field of the data-type, to evaluate the information that could be useful to the specific problem. None of the authors can be considered experts of EEG data, so most of the different proposed features in this thesis are based on previous research within EEG- and EOG-data. The choice of some of these, such as the Pearson Coefficients [60], will be argued for in Section 4.1.2 as part of this work. Some of the proposed features was however found through studies of the eye movement plots. These features were found through work in this thesis and is listed as follows:

- Standard deviation
- Maximum value
- Minimum value
- Difference in min value between two channels
- Difference in max value between two channels
- Peak-to-Peak (with inspiration from [61])

4.1.2 Proposed features

Through the literature search and some thought experiments, a list of features was thought to be able to classify the eye movements. The proposed features are presented in this section.

Samples as features

Using the sampled values, where each sample is considered a feature is a good approach in many scenarios. Not only can it provide the best classifier performance, but it can also be used to compare performance between the original data and data represented by abstracted features [33]. The samples make up the original signal $x[n]$ and should in some way contain the information that is needed to distinguish the different classes.

Following the rule of thumb where the number of datapoints should be five to ten times larger than the dimensionality of the feature vector per class [38], there is too little data in the datasets made in this work to use this approach. As each movement is represented by 250 samples, the suggested amount of datapoints would be $1250 - 2500$ for each class. This is time consuming to make, and a difficulty to overcome when using BCI technology with machine learning. Another approach would be to reduce the window size (feature vector) representing the eye movements and see how the results turns out. If the window size (feature vector) was at 50 samples, the suggested number of datapoints needed for training would be at $250 - 500$ which is within reach given the size of the datasets in this work, as given in Table 3.5.

Higuchi Fractal Dimension

Fractal dimensions are measures of the complexity of the recorded signals and is often used on EEG data [62]. Non-linear analysis of EEG data through fractal modeling can give a deeper understanding of underlying physical processes that is involved in EEG [63]. The Higuchi Fractal Dimension (HFD) algorithm [64] approximates the mean length of fixed size curve segments and has been used in sleep stage research [65, 66]. The calculation procedure can be found in Appendix C.2

The calculation of HFD was done with PyEEG, a Python module designed for EEG feature calculation [67]. Two HFD features are calculated, one for a channel suited for vertical movements ($x_1[n]$) and one for a channel suited for horizontal movements ($x_4[n]$).

Petrosian Fractal Dimension

Another fractal dimension-based feature used is the Petrosian Fractal Dimension (PFD). The reason for considering this feature is that it was provided in the PyEEG module. Not much has been found that supports this feature in an eye-classification scheme, but it is taken into consideration through introduction from the PyEEG module. There are in total two PFD features, extracted from channel $x_1[n]$ and $x_4[n]$ respectively. The formula for the PFD can be found in Appendix C.

Standard deviation

Standard Deviation (STD) is a measure of variability [68]. There is a difference in the variability of some of the movements, such as looking straight ahead, blinking and looking to the left. This difference can be illustrated in Figure 4.1.

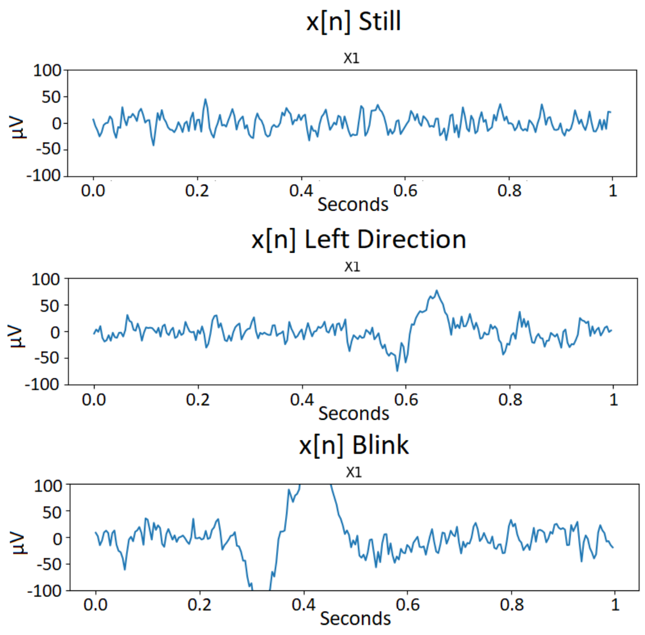


Figure 4.1: Illustration of varying STD between looking straight ahead, left and blinking.

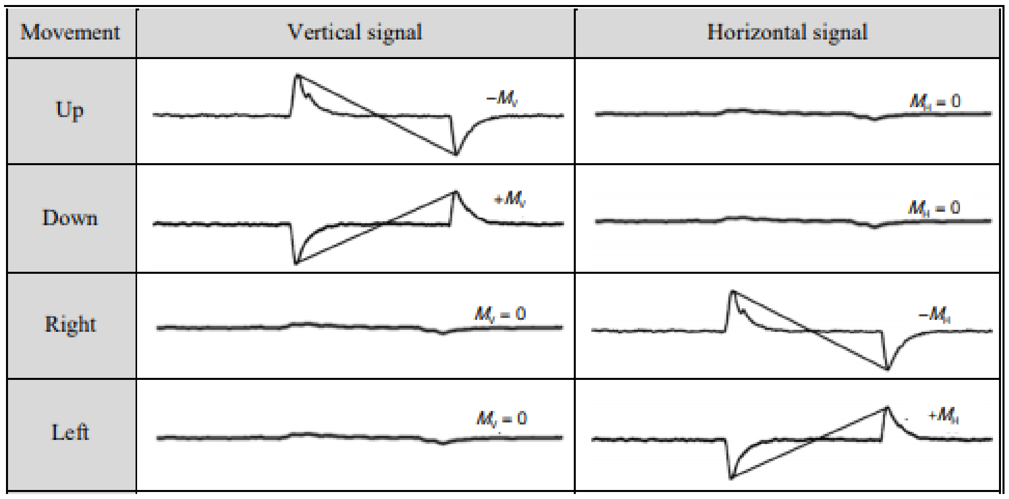
The eye movement plots in Figure 4.1 was gathered from experiments performed in this work, where a subject looks in different directions and the movements are plotted. As shown in Figure 4.1, the STD will be low when looking straight ahead, and will increase when performing horizontal and vertical movements. The STD increases further for blinks as a result from the increasing minimum and maximum signal values. STD is therefore considered to be a strong feature to tell the difference between looking straight ahead, horizontal or vertical movements and blinking. The STD is calculated using the function "std" provided by Numpy [69]. Two STD features are calculated for channels $x_1[n]$ and $x_4[n]$ respectively.

Maximum and minimum values

Maximum and minimum amplitudes can be used to distinguish looking straight ahead, blinks and horizontal or vertical movements from each other by looking at channel 1. Figure 4.1 from the STD feature shows plots of the different movements, and there is a difference in the signal-levels between the three types of movement. Looking straight ahead has almost no variation and will therefore have low min and max values. Horizontal and vertical movements have around the same min and max values, while blinks have the largest min and max values. The features are calculated with the "min" and "max" functions provided by Numpy [69].

Slope between maximum and minimum points

The slope feature was chosen through presented results in [61], which proposed to use the slope between min and max points to classify different eye movements. The feature was originally used in EOG recordings and show that the slope in channels $(x_1[n] \wedge x_2[n])$ and $(x_3[n] \wedge x_4[n])$ differ when making specific movements. Figure 4.2 shows a table from [61] which illustrates how the slope differs between movements.



Movement	Vertical signal	Horizontal signal
Up	$-M_V$	$M_H = 0$
Down	$+M_V$	$M_H = 0$
Right	$M_V = 0$	$-M_H$
Left	$M_V = 0$	$+M_H$

Figure 4.2: Illustration of different slopes from different movements [61].

The feature is utilized in the same way in this work but is expected to have varying importance due to variation in the signals as shown in 3.12. Something to notice is that the table in Figure 4.2 [61] shows that the slope is used with a pair of two eye movements (in a direction and back again), whereas it was used on one eye movement only in this work. There are two slope features calculated, one for a channel suited for vertical movements ($x_1[n]$) and one for the horizontal movements ($x_4[n]$). The slope S is defined by Equation 4.1 [61].

$$S = \frac{SV_{min} - SV_{max}}{P_{min} - P_{max}}, \tag{4.1}$$

where:

- SV_{min} = minimum signal value
- SV_{max} = maximum signal value
- P_{min} = position of minimum signal value
- P_{max} = position of maximum signal value

Peak-to-Peak

Peak-to-Peak (PTP) calculates the range of values, meaning the difference between the maximum and minimum points within the window $l_{w,c}$ [69]. This feature is based on much of the same argumentation as the slope feature but does not consider the distance between the maximum and minimum value indexes. The PTP feature was chosen through inspiration from the slope feature in [61]. The calculation is done with the function "ptp" from Numpy and is equivalent to taking "max" - "min" as provided by Numpy [69]. There are in total two PTP features that are calculated for channel x_1 and channel x_4 (vertical and horizontal).

Difference in minimum and maximum point between two channels

As it can be seen in Figure 4.3 there are differences in the maximum and minimal points in channels $x_2[n]$ and $x_4[n]$ when performing some of the movements. This was observed through experiments in this work, where a subject looked in different directions. The green squares mark the maximum point, and the red squares marks the minimum point in the respective channels.

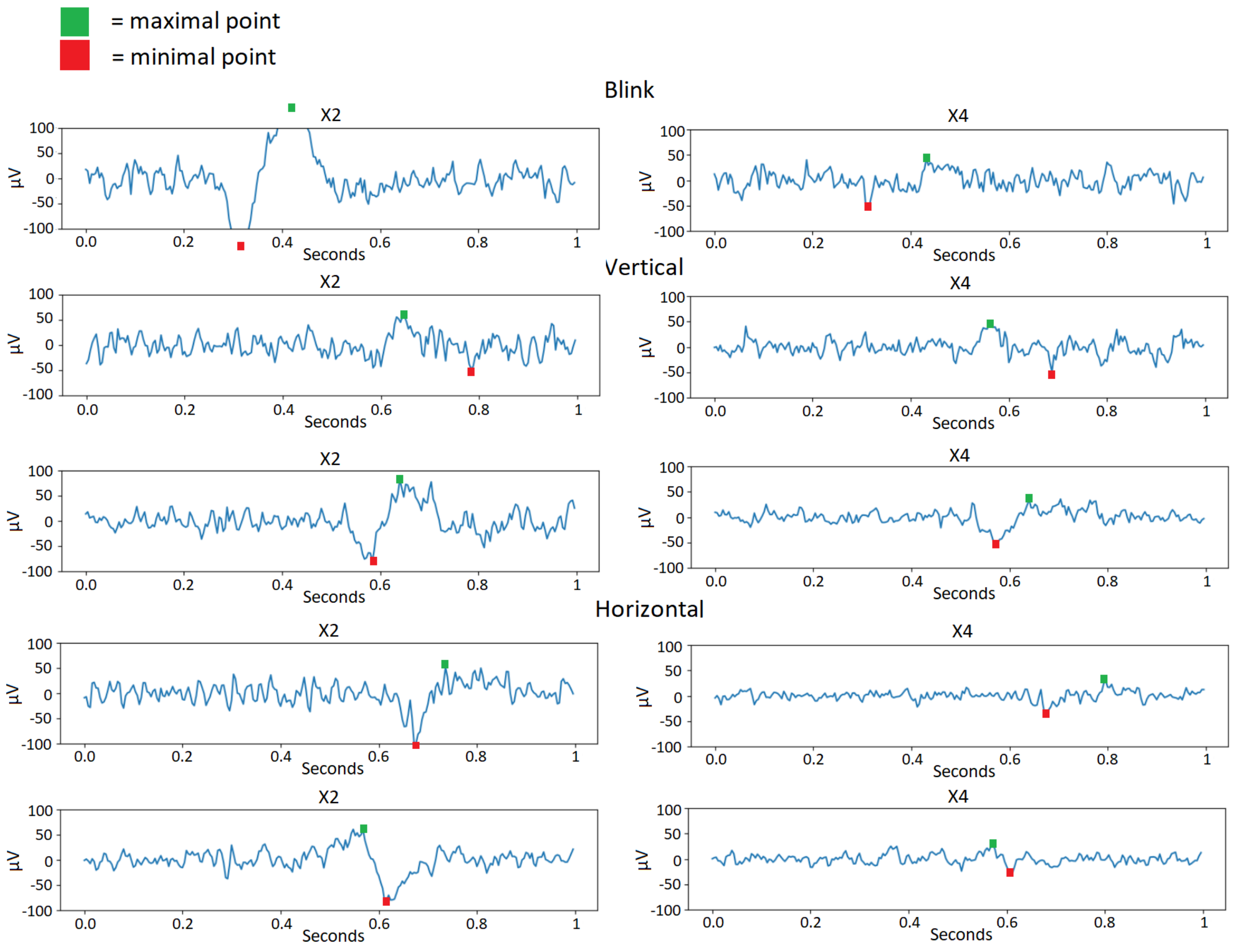


Figure 4.3: Illustration of how the vertical movements, horizontal movements and blinks differ in maximal and minimal points between channel 2 and 4.

The features are defined by Equation 4.2 and 4.3.

$$\text{diff}_{min} = \min(x_2[n]) - \min(x_4[n]) \tag{4.2}$$

$$\text{diff}_{max} = \max(x_2[n]) - \max(x_4[n]) \tag{4.3}$$

This is done with the "max" and "min" functions provided by Numpy [69].

Covariance between two channels

The covariance feature describes how the samples within the window of size $l_{w,c}$ varies between two channels. It is thought that different channels will vary differently according to the movements performed. This will be illustrated shortly when presenting the Pearson Correlation features. There are in total 4 covariance features used for channel pairs: $(x_1[n] \wedge x_4[n])$ and $(x_3[n] \wedge x_4[n])$. Each channel pair is represented by two covariance features. The calculation is done with the "cov" function provided by Numpy [69], and can be found in Appendix C.

Pearson Correlation Coefficients

Pearson correlation is the most widely used type of correlation and is also called linear or product-moment correlation [60]. The Pearson coefficient is a measure of how linearly related two variables are and has been used with EEG data to tell if a subject has open or closed eyes [70]. The calculation procedure can be found in Appendix C.3. The coefficient has a high value if the relationship between the variables can be approximated by a straight line.

The idea is that the correlation between two channels is different when performing different movements. For this, channel pairs $(x_1[n] \wedge x_4[n])$ and $(x_3[n] \wedge x_4[n])$ are used. This is because channels $x_3[n]$ and $x_4[n]$ seems to be similarly "active" when performing horizontal movements, blinks and left movements. The same channels seem to differ in "activity" when performing the right movements (right direction and returning from left), where channel $x_4[n]$ has distinct variations while $x_3[n]$ seems to not be that affected by the movement. This was observed through experiments in this work, where a subject looked in different directions. The similarities from the experiments of horizontal movements, blinks and the left movement are shown in Figure 4.4.

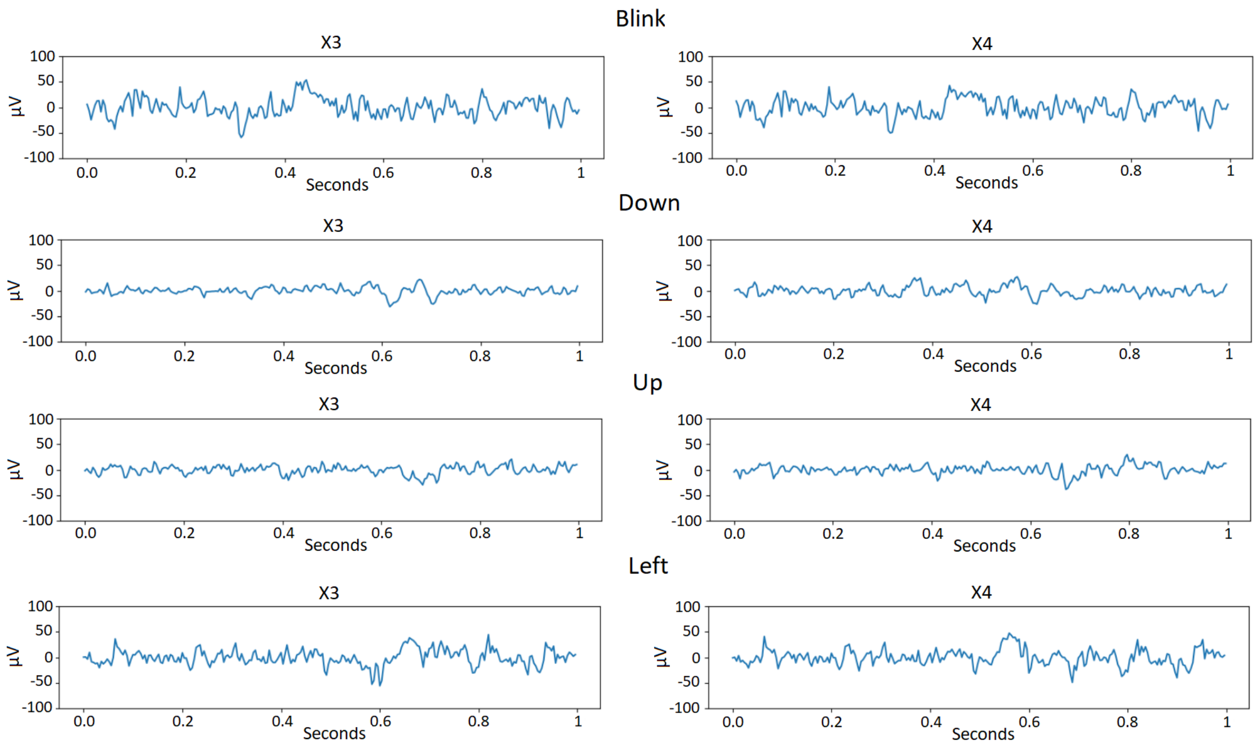


Figure 4.4: Illustration of how the blinks, down, up and left movements seem similarly "active" in channels 3 and 4.

The difference in channel $x_3[n]$ and $x_4[n]$ when performing right movements is shown in Figure 4.5.

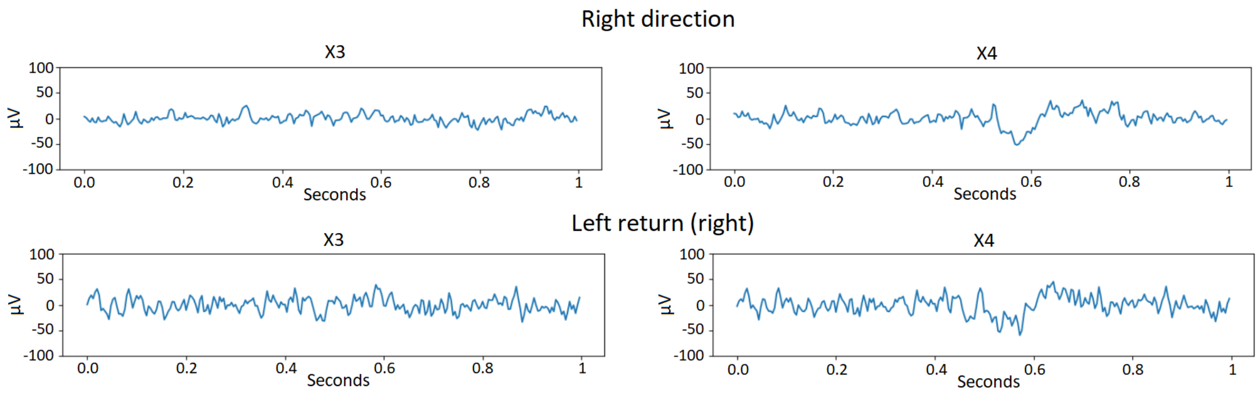


Figure 4.5: Illustration of how right movements differs between channel 3 and 4.

Figure 4.6 shows how there are noticeable changes in both channel $x_1[n]$ and $x_4[n]$ when performing horizontal movements, but while performing vertical movements channel $x_1[n]$ seems to be the channel that is mostly affected.

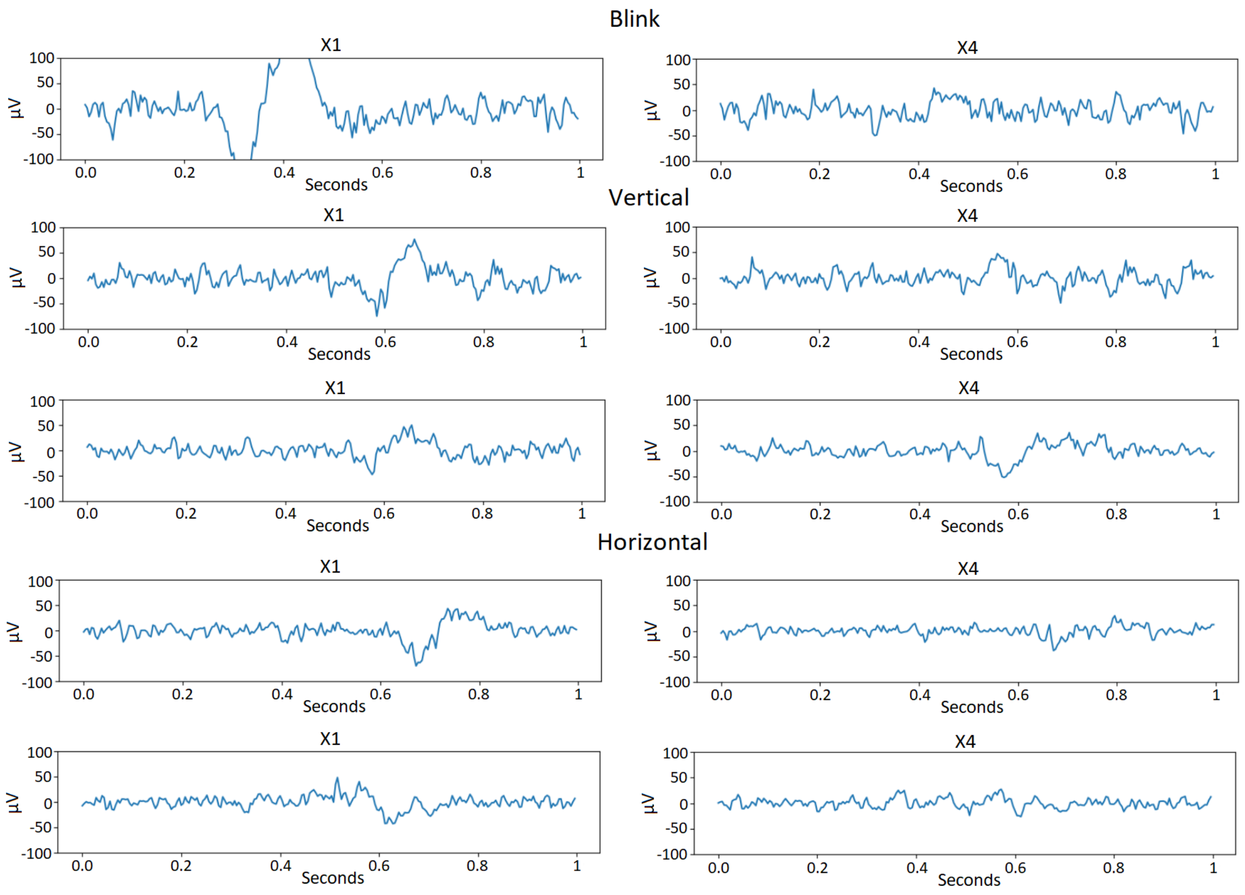


Figure 4.6: Illustration of how the vertical movements, horizontal movements and blinks differ in the channel pair 1 and 4.

The calculation of the Pearson Correlation Coefficients is done using the "corrcoeff" function provided by Numpy [69]. Two of the four calculated coefficients are utilized, meaning that the Pearson coefficients are represented as a total of 4 features. This is because a 2×2 matrix y is returned from the Numpy function, where the elements $y_{1,1}$ and $y_{2,2}$ has the value 1, while $y_{1,2}$ and $y_{2,1}$ describe the value of the correlation.

Theta-beta band ratio

[41] showed that using features based on spectral band properties can provide good results. They achieved an average eye movement classification accuracy of 90.1% using features they called the Theta-Beta ratio (TBR). TBR is the ratio between average amplitude values in the brain wave frequency bands Theta and Beta from Table 2.1. The formula for TBR can be found in Appendix C. Other features were also proposed in [41] but was shown to give little value. There are in total two TBR features used in this work, calculated from channels $x_1[n]$ and $x_4[n]$ respectively.

Power Spectral Entropy

Power Spectral Entropy (PSE) has been used as an EEG feature in research of imaginary motor tasks and anaesthetic state classification, with good results in [71, 72]. Entropy is a measure of complexity and is regularly used in EEG research [62]. There are many types of entropy available, but the choice came upon PSE [73] because of the the proven results in [71, 72]. As with HFD, PSE is supported by the PyEEG module [67] and is utilized to calculate the feature. Two features are calculated, one for a channel suited for vertical movements ($x_1[n]$) and one for horizontal movements ($x_4[n]$). The procedure for calculating PSE can be found in Appendix C.1.

4.1.3 Quality of features

The quality of a feature can differ. Some features might give very distinct differences in the data provided, while other features might be irrelevant and even redundant. To make sure that the most important features are used in this work, among those proposed, a system to test the classifier performance based on the selected features is used. Something that is important to understand when evaluating the quality of features, is that a feature can be irrelevant by itself but can become relevant when used in a combination [33]. A subset of features is a combination of features based on a set of proposed features, such as shown in Figure 4.7.

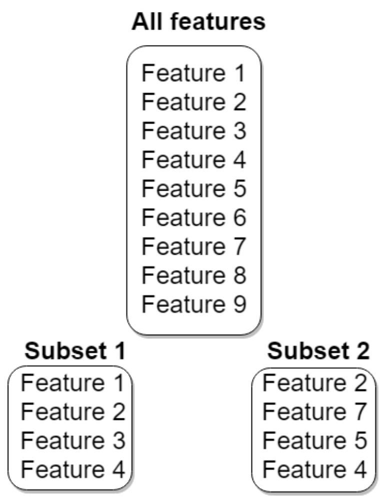


Figure 4.7: Illustration of different subsets of features.

It is possible for Feature 2 from Figure 4.7 to have a low impact in subset 1, but to have a high impact in subset 2. This is because some features may have high correlation, making it hard for the model to differentiate between the classes. High correlation between two features can serve as noise and can reduce the performance of the classifier [74]. In another case, where the features have distinct different relevant properties, Feature 2 can have a higher impact because it provides additional information to the other features selected. Finding the perfect subset (partition) is a computationally expensive task, as the best subset might only be found by doing a complete search between all the possible combinations of features and compare the performance of the classifier for each combination. Instead of performing a complete search of every combination, there are methods to find a good subset of features often used in machine learning. The method utilized in this work is called the Recursive Feature Elimination with cross-validation (RFECV) and will be explained in Section 4.1.4. Choosing a subset of features for the task is called feature selection and uses the proposed set of features as a starting point.

4.1.4 Feature selection

Feature selection helps in reducing the computational requirement and the complexity of the classification task, by focusing on selecting a subset of features which efficiently describes the input data. Finding those features that are usable for the classification task, is the key focus of feature selection [33]. If a classifier uses irrelevant features during training, this irrelevant information will be used on new data during testing as well, leading to a poor generalization [74]. There are many ways to decide what feature subset one should use, as different optimization algorithms work in different ways to find the subset of features giving the best results.

There are three types of feature selection methods: Filter, Wrapper and Embedded selection methods [74]. *Filter* methods are used as a pre-step to evaluate the quality of the features before training the classifier. A suitable ranking criterion is used to score the features, and a threshold is chosen to remove features of low relevance [74].

Wrapper methods uses the classifier as a black-box and looks at how the features chosen impacts the performance (accuracy for example) of the classifier by testing it. This means that a classifier is trained for each subset of features. An extensive search can become computationally intensive for larger datasets, where larger feature vector dimensions also increase the amount of computation. Wrapper methods are therefore used with simplified algorithms such as Genetic Algorithms [74].

Embedded methods try to reduce the computation time taken up for reclassifying different subsets, which is done in wrapper methods. The main focus is to incorporate feature selection during the training process [74]. Recursive Feature Elimination (RFE) is a standard embedded method for doing feature selection with SVMs, which uses the importance of features to decide feature removal [75]. A *weight* is realized by the classifier during training and is based on how the classifier uses a feature when making predictions, each feature gets a weight. The weight can therefore be looked at as the importance of a specific feature. The RFE function takes in the feature importance in form of the feature weights, the higher value of the weight the more important the feature.

Many feature selection methods are considered unstable, meaning that they do not produce the same feature subset when altering the train and test split. Therefore, to make sure that the method used is stable, several train and test splits must be created. This is to prove that the same features and the number of features is the same for the different splits.

RFE is a recursive greedy algorithm. It starts on the full set of features, considers all the weights and removes the feature with the lowest weight. This is done in several iterations, training a classifier for each iteration, until the desired number of features is reached. This method leaves out some feature combinations as the feature left out never will be a member of a considered feature subset again. Another approach utilizes something called Cross-validation (CV), which will be explained in Section 4.3.2, to calculate the performance of the classifier at each feature subset. This is called Recursive Feature Elimination with cross-validation (RFECV) and is provided by Scikit-learn [76]. RFECV outputs the number of features which gave the best performance of the classifier, and what features were selected to achieve that performance. Figure 4.8 shows how an example plot of how classifier performance is evaluated at each step when using RFECV.

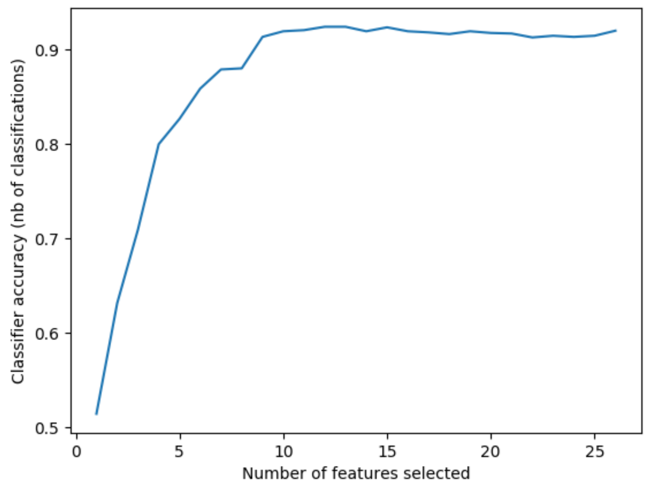


Figure 4.8: Example plot showing how RFECV evaluates the performance at the different steps in its process [76].

RFECV is used in this work for feature selection and is chosen because of the easy implementation as provided by Scikit-learn [76] and because it is a standard feature selection method to use with Support Vector Machine (SVM)s [75]. As will be shown later, SVM is the machine learning algorithm of focus in this thesis and will be explained in more detail in Section 4.4.1.

4.2 Scaling and splitting the dataset

So far, the raw data $\tilde{x}[n]$ has been processed through filtering and features has been extracted. Before one can feed $\mathbf{f}_v[i]$ to the algorithm it might be necessary to scale the data [77], and to split it up into data used for training and for testing.

4.2.1 Scaling

Algorithms such as Support Vector Machine (SVM) and K-Nearest Neighbors (KNN) require features to be standardized, meaning that they have zero mean and unit variance [77]. If a feature has variance that is magnitudes times larger than the other features, it will have a much larger impact (larger weight) than the others, and make the classifier interpret the significance of the features incorrectly. This is because of how the classifier interprets the magnitude of feature values during training. Each feature is therefore standardized to have zero mean and unit variance, in order to allow the classifier to learn evenly from all features. This is done by training a scaler through calculating the mean and standard deviation from the training data with the "StandardScaler" function provided by Scitkit learn [78]. These mean values and standard deviations collected from the training data is then used on the test data, so the scaler is also realized through training.

4.2.2 Training and test data

An important step before feeding data to a classifier is to split the dataset up in two parts which is used in separate steps as illustrated in Section 2.5. These two parts are called training and testing. The training data is as used to train the classifier, while the test data is used to validate the performance of the classifier with a test. If the same data was used for both training and testing, there would be no way to tell if the classifier is predicting correctly, or if it is copying what it was given in the training stage. The dataset is split into training and testing with a ratio of 80/20 throughout this work.

The test part is only used once in the end to validate the performance of the classifier. When splitting the data into training and testing, it is important to understand that the performance of the model will have some variation if the data contained in each split change. In practice, methods splitting the data often use some kind of randomizer to make sure that different splits are produced, which makes for a more unbiased evaluation. More about why splitting the data into training and testing is so important will be explained shortly.

Balanced training and test data

The training and testing sets are sorted so that the number of datapoints for the different classes are balanced. This means that there is an equal proportion of each class in the training data, and in the test data. This ensures that the classifier is not biased towards some of the classes, as a larger amount of one class in the training data can make the classifier biased towards that specific class, as mentioned in Section 3.3.5.

4.3 Machine learning algorithms, models and evaluation

There is a difference between a machine learning algorithm and a machine learning model. A machine learning algorithm tries to solve a specific problem in a set of ways when being fed with data. Support Vector Machine (SVM), Decision Tree, K-Nearest Neighbors (KNN), Naive Bayes and Logistic Regression are examples of common and popular algorithms [79], all working in different ways to distinguish between the data. These algorithms have something called *hyper-parameters*, which are parameters that cannot be learned through training, but are set before training is started.

The hyperparameters can represent many different properties, such as how the classifier learns from mistakes, how it should shape the decision boundary according to the training data, how strict it should be when making predictions etc. A model is specific to the hyperparameters chosen and training data, meaning that there are many different models that can be created from each algorithm. Therefore, when talking about training and testing, it is the model (classifier) that is being trained and tested. Tuning the hyperparameters is an important part of designing a good classifier [80]. How these hyperparameters are tuned is explained in Section 4.4.2.

Choosing the correct algorithm for the task can be difficult. Scikit-learn has provided a cheat-sheet shown in Figure 4.9, which is designed to give a starting point when choosing an algorithm.

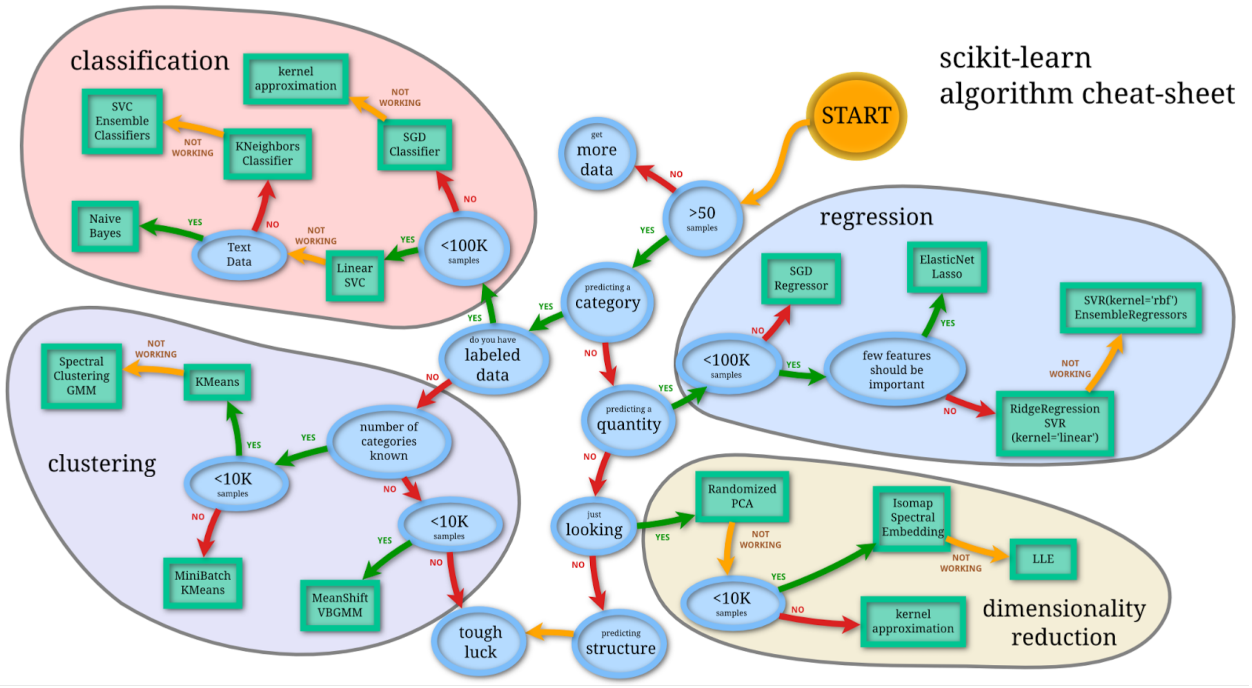


Figure 4.9: Map of proposed method for choosing a machine learning algorithm, provided by Scikit Learn [81].

In Figure 4.9 the term "samples" is equivalent to datapoints, which is the term used in this thesis. There is no algorithm that works best on a general basis, which means that the choice can depend on factors such as the problem trying to be solved, number of datapoints, quality of data, the nature of the data, the features representing the data etc. There is therefore no right answer to the question "Which machine learning algorithm should I use?" [36]. This also means that there are several algorithms that can solve the same problem. Machine learning is therefore often subject to an experimental approach, trying to implement with one algorithm and move on to another if the results are insufficient.

4.3.1 Scoring metrics

Evaluating the classifier is important to say something about the performance. The system presented in this thesis tries to differentiate between six classes related to eye movements. The predictions from the classifier are listed in a *Confusion Matrix*. Figure 4.10 shows the Confusion Matrix of a test performed early in this work where a classifier was trained on a training set with 119 datapoints, 20 for each class with an exception of blinking which has 19 datapoints.

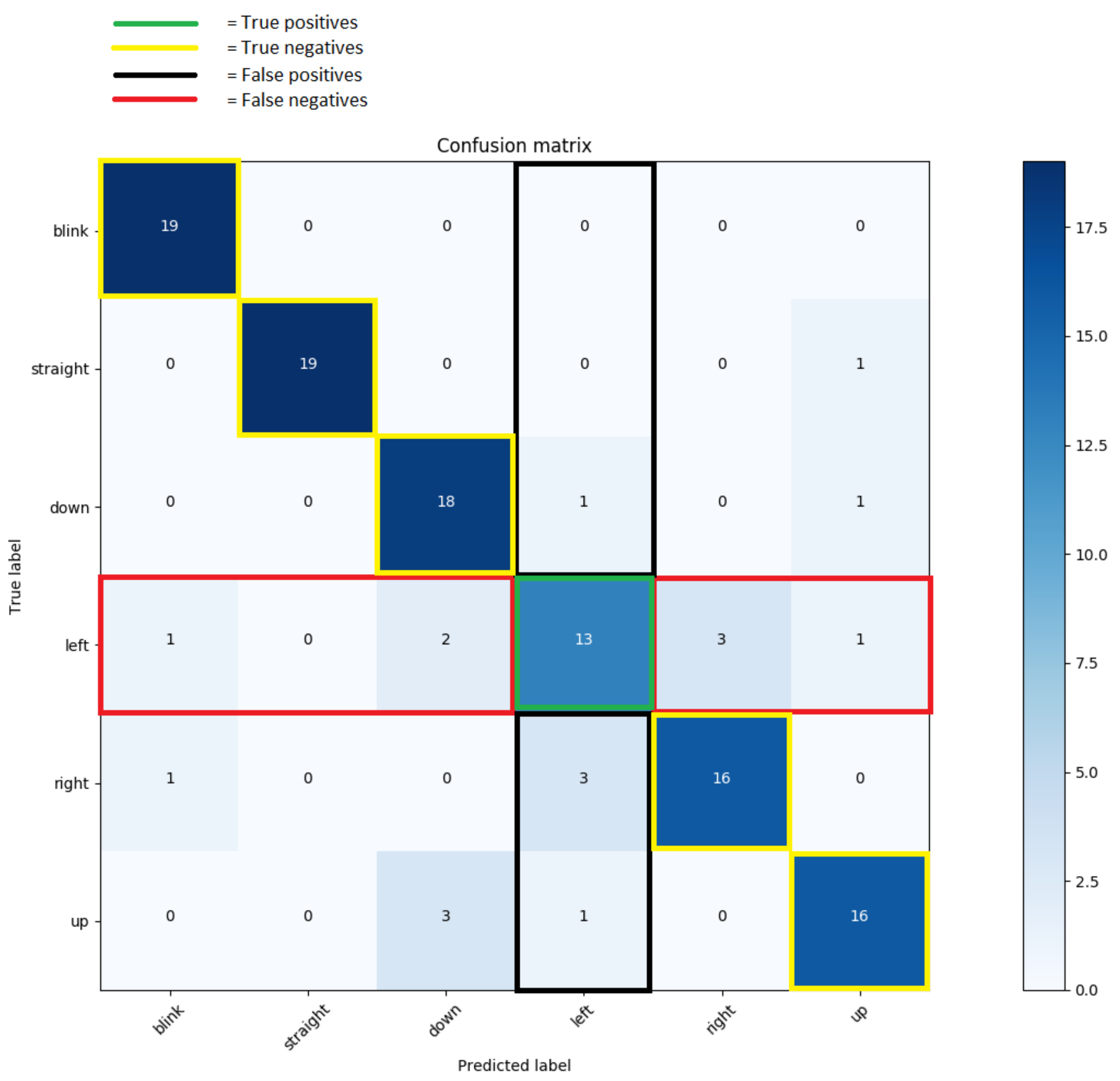


Figure 4.10: Illustration of a Confusion Matrix used to evaluate the Left class. The matrix is based on a test set with 19 datapoints for blinking, and 20 for the rest. The layout of the Confusion Matrix is provided by Scikit-learn [82].

The rows in a Confusion Matrix show what the classifier should have predicted, and the number in each cell shows how many times that specific prediction was made. The columns show the actual predictions made by the classifier. Therefore, the diagonal cells show how many times the classifier predicted correctly.

The Confusion Matrix gives a very intuitive understanding of what the classifier does right and what it does wrong. There are several scoring metrics that can be used to evaluate the classifier, some metrics may be more important for some systems than others, and some can be prone to weaknesses. The scoring metrics are based on the following terms:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

When using these terms, it is important to note that the metrics are calculated for one specific class at a time. To get the overall performance of the classifier for a given metric one must first find the metric for all the respective classes, then average the score to get total performance. When calculating the metrics respectively, the class that is under focus is considered the positive class, while all others are considered to be the negative class. With this information, we can define the terms introduced:

- *True positives* - should predict the positive class and actually predicted it
- *True negatives* - should predict the negative class, and actually predicted it
- *False positives* - should predict the negative class, but predicted the positive class
- *False negatives* - should predict the positive class, but predicted the negative class

Using these terms while looking at the Confusion Matrix in Figure 4.10 makes them easier to understand. Let the Left class be of focus, it is then considered as the positive class while all others are considered to be the negative class [83]. The vertical axis shows the true classes, while the horizontal axis shows the actual predictions made by the classifier. Therefore, True Positives are found in the diagonal square marked in green, False Negatives are found in the horizontal squares marked in red, False Positives are found in the vertical squares marked in black and True Negatives are found in the diagonal squares marked in yellow.

Confusion matrices are often used to evaluate performance and get a visual feel of what the classifier does wrong and correct. The most intuitive and proper evaluating is done through analyzing the Confusion Matrix, as all the classifier's weaknesses and strengths are shown. This is the basis of the scoring metrics, where the number of true positives, true negatives, false positives and false negatives are all counted from the Confusion Matrix. A few scoring metrics that are frequently used to evaluate the performance are defined as follows:

Accuracy

Accuracy A is the proportion of correct classifications from overall number of classifications, defined by Equation 4.4 [83].

$$A = \frac{\mathrm{TN} + \mathrm{TP}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{4.4}$$

Precision

Precision P is the proportion of correct positive classifications from cases that are predicted as positive, defined by Equation 4.5 [83].

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{4.5}$$

This says something about how many of the positive class predictions actually were correct. Precision is also called the positive predictive value [83].

Recall

Recall R is the proportion of correct positive classifications and the total number of the positive class and is defined by Equation 4.6 [83].

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{4.6}$$

This shows how many of the positive class it predicted correctly, based on how many it should have predicted correctly. Recall is also called sensitivity [83].

f1 score

The f1 scoring metric is the harmonic mean of the precision P and recall R, defined by Equation 4.7 [83].

$$\text{f1} = \frac{2PR}{P+R} \tag{4.7}$$

Which also can be defined by Equation 4.8.

$$\text{f1} = \frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}} \tag{4.8}$$

Scoring example and Classification Report

This example is created in this work to illustrate how the presented scoring metrics can be used, and how some of them can have weaknesses.

Using Figure 4.10 and the Left class as an example we can calculate some performance metrics. The number of True Positives is found in the diagonal square, with a total of 13 True Positives. Looking at the vertical squares for the Left class, it is seen that there were Left predictions where Down, Right and Up were the true classes. In total there were $1+3+1=5$ False Positives for the Left class. Looking at the horizontal squares we can see that there were left movements classified as Blink, Down, Right and Up. Giving a total number of: $1+2+3+1=7$ False Negatives. The number of True Negatives for the Left class is equal to the sum of True Positives from the other classes. Thus, summing the diagonal squares and leaving out the Left class gives TNs = $19+19+18+16+16=88$.

Now that the parameters are gathered, the scoring metrics can be calculated.

Accuracy:

$$A = \frac{\text{TN}+\text{TP}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$$

$$A_{\text{Left}} = \frac{88+13}{13+88+5+7} = 0.89$$

This seems like a good result as the accuracy is high, but this illustrates how the scores can be deceiving. In this case, it is deceiving because of the fact that there are a lot more predictions from the other classes, as the Left class only stands for $\frac{1}{6}$ of the total predictions. So, the fact that there are a lot of True Negative boosts the accuracy metric.

Precision:

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$P_{\text{Left}} = \frac{13}{13+5} = 0.72$$

The precision score shows that the predicted Left class was correct in 72% of the cases it was predicted. Leaving 28% of the Left predictions as wrong.

Recall:

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$R_{\text{Left}} = \frac{13}{13+7} = 0.65$$

This recall result shows that 65% of the Left class datapoints from the test set were actually predicted as the Left class. Leaving 35% of the Left class datapoints predicted as something else.

From this example it can be seen that several metrics often need to be considered to evaluate the classifier. In the case of evaluating the performance of the classifier when looking at classes respectively, in a multiclass-scheme, accuracy seems like a bad point to start. Accuracy could be better suited when looking at the overall prediction when considering all classes. To the authors, a balance between precision and recall seems like a good evaluation point when looking at the classes respectively. This is because a low recall score for one class would decrease the precision for another class, where the two metrics combined both considers FP and FN. While scoring metrics such as accuracy only considers the hit rate. For optimal evaluation, the Confusion Matrix should be studied to show the weaknesses and strengths of all classes.

When presenting the scoring metrics in this thesis, they will be split in two parts. The first part considers all the classes and calculates the scores as an unweighted "Macro-average" [84], meaning that the number of datapoints for the respective classes used in the test is not considered. This will be referred to as the "unweighted average".

The second part presents scores for the respective classes, and a weighted "Macro-average" score [84], meaning that the number of datapoints for the respective classes affects the score. The weighted "Macro-average" of a scoring metric is considered as the "weighted average" throughout this thesis. A "Classification Report", which is created with a function provided by Scikit-learn [85], lists both the respective class metrics and the weighted average. An example of a Classification Report can be seen in Table 4.1 [85]. The two ways of presenting the scoring metrics will be shown as follows.

First part: The model achieved an accuracy of 78.8%, precision of 78.8% and recall of 81.0%. These are the unweighted average scores.

Second Part:

Table 4.1: Classification Report

Class	Precision	Recall	f1-score	support
Blink	0.78	0.54	0.64	13
Down	0.40	0.80	0.53	5
Left	1.00	0.83	0.91	6
Straight	0.86	0.83	0.85	30
Right	0.83	1.00	0.91	5
Up	0.86	0.86	0.86	7
avg/total	0.82	0.79	0.93	66

Table 4.1 shows the metrics for each class and then a weighted average score for all the classes at the bottom line. The support column shows how many datapoints that was tested for each class.

To show how the weighted and unweighted average scores are calculated, we can use the "Classification Report" in Table 4.1 as an example. We can start by calculating the weighted average scoring metric for Precision. This is done by summing the Precision scores for the respective classes and divide by the number of classes as in Equation 4.9.

$$\frac{0.78+0.40+1.00+0.86+0.83+0.86}{6} = 0.788 \tag{4.9}$$

The weighted average scores, as can be found as the "avg/total" in the Classification Report, uses the same respective class scores as presented in the Classification Report but finds the weighted average. The support column is used to weight the scores as in Equation 4.10.

$$\frac{0.78 \times 13 + 0.40 \times 5 + 1.00 \times 6 + 0.86 \times 30 + 0.83 \times 5 + 0.86 \times 7}{66} = 0.82 \tag{4.10}$$

The unweighted and weighted average scores are as shown, the same as presented in the "First part" and at the bottom of the Classification Report.

4.3.2 Evaluating model performance

Splitting the dataset into training and test sets is an important step before a specific model is realized through training and tuning of hyperparameters, as mentioned in the beginning of Section 4.3. If a model were to be trained on a given dataset, and was tested on the exact same data, there is no way to tell if the model actually predicted something useful or just repeated it [86]. This concept where the test data is too correlated with the training data is the reason for splitting up in training and test data and is common practice in machine learning to prevent what is called *overfitting* [86]. Figure 4.11 illustrates how overfitting fails to generalize the model.

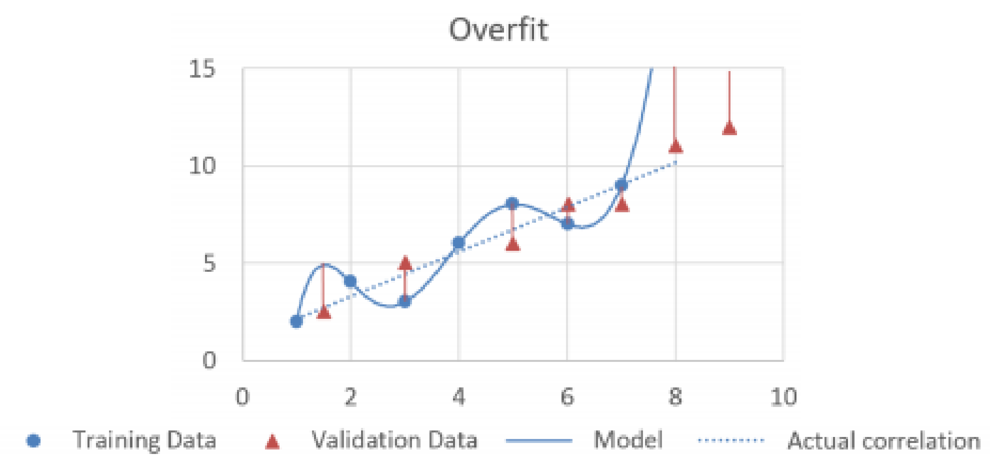


Figure 4.11: Illustration of overfitting [87].

Overfitting describes the situation where a model that fits the training data can have a lot of correct predictions but fails to predict anything useful when presented with new data not previously seen during training [88]. *Underfitting* is the opposite of overfitting, where the model will generalize too much and fail to find a decision boundary which can perfectly classify the datapoints [89]. Even when introducing the splitting of data into training and test data, there is a chance of overfitting. This is because of the fact that the performance can be optimized for the test set by tuning the hyperparameters. Therefore, one could say that information specific to the test set leaks into the trained model [86]. There are two ways to avoid this. The first is to split the dataset into yet another part called the validation set. This validation set will then be used as evaluation when tuning the hyperparameters, and when things seem to be going smoothly, the test set can be used for final evaluation [86]. A problem with this is the decrease in data that can be used for training. The second way is called Cross-validation (CV). A test set should still be held out when using CV, but there is no need for a validation set [86]. Figure 4.12 shows the principle of Cross-validation.

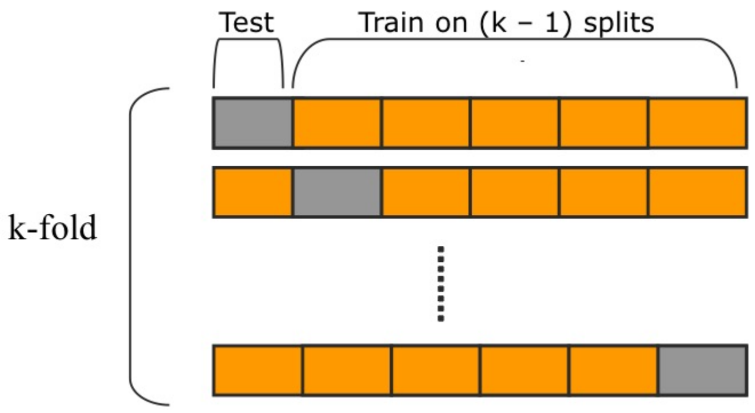


Figure 4.12: Illustration of the how cross validation is performed [90].

Cross-validation splits the training data into k different folds, where $k-1$ folds are used for training and the last fold is used for validation. This is repeated k-times, where the fold used for validation changes every iteration and ensures that no fold is used for validation more than once. This prevents that the hyperparameters are biased (overfitted) towards specific test data [86]. It is important to note that CV only uses the training data from the split of the dataset and does nothing with the test set. CV can be used for both tuning of parameters and evaluation of a model [86]. When using CV for evaluation, a specified scoring metric is calculated for each iteration and appended to a list. It is possible to show the scoring metric from all the different iterations and to output the average and standard deviation of the scores. Evaluation with CV is performed on the training set, and then compared with results from the test set. Big differences in the scores from the Cross-validation and test set can indicate overfitting and underfitting.

4.4 Classification scheme

As mentioned in Section 2.5, there are a lot of viable algorithms that can be used, and there is no algorithm that always will outperform the others. Support Vector Machine (SVM) is the main focus and was chosen because of good results and reviews for EEG data and pattern recognition in literature [37, 39, 43, 91, 92, 93, 94].

4.4.1 Support Vector Machines

SVM is a powerful approach for pattern recognition. It does not need any distributional assumptions about the data, while providing a very good discriminative solution and generalization at the same time [94]. SVM classifies new datapoints with a defined optimal hyperplane (decision boundary). The hyperplane is realized through the training phase and exists in a high-dimensional space [95]. For illustration purposes, one can think of the hyperplane in a two-dimensional space where it simply equals a straight line as illustrated in Figure 4.13.

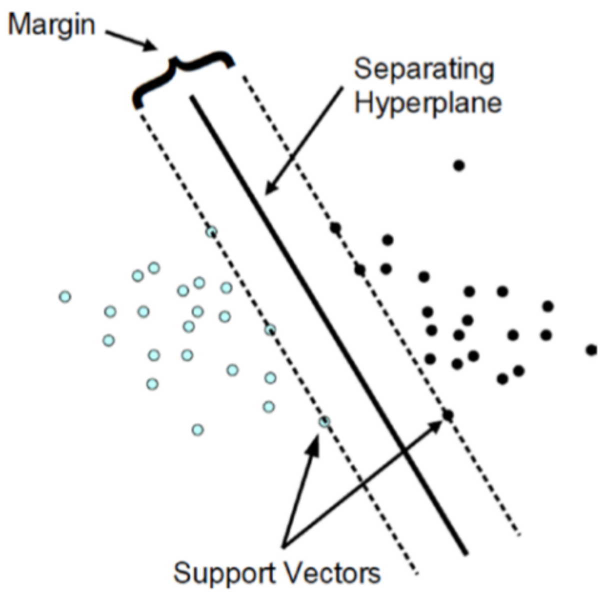


Figure 4.13: Illustration of the hyperplane used to discriminate between classes [96].

Looking at Figure 4.13 there are a lot of ways one could draw a line to separate the two classes. The problem the SVM therefore tries to solve is to find the hyperplane with the largest possible margin between distributed datapoints [95]. The margin is defined as the distance between the hyperplane and the closest datapoints. Intuitively enough it becomes easier to generalize with a larger gap (margin) between the classes. Support vectors are parallel vectors of the hyperplane, which is defined by the datapoints on the margin [95]. Figure 4.13 showed an example of a linearly separable problem where a linear SVM discriminates between the classes. But what if the problem is not linearly separable with the given datapoints?

To solve this problem, many machine learning algorithms (including SVM) uses something called a *kernel-transformation*. The idea is that if the data is not linearly separable in the current dimensional space, it may be transformed into a space called a *kernel space* where the data is linearly separable by a hyperplane [95]. Figure 4.14 illustrates a simplified kernel-transformation [97].

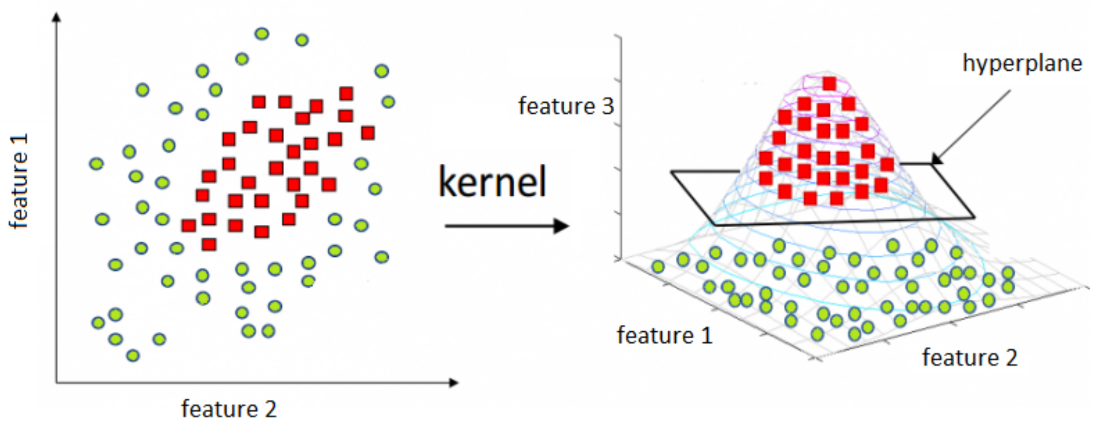


Figure 4.14: Illustration of the kernel transformation used to find a linearly separable hyperplane [97].

The transformation is done with a given *kernel function*. There are many kernel functions that can be used, those supported by Scikit-learn for SVMs are Radial Basis Function (RBF), linear, Polynomial and Sigmoid kernels. As with machine learning algorithms, there is no set science in determining SVM kernels. According to the no-free-lunch theorem [98] there are no guarantees that one kernel will work better than the other. In this work the RBF kernel and linear kernel were chosen due to RBF kernel popularity and the simple nature of a linear kernel [99]. Table 4.2 shows the definition of the different kernel functions for SVMs, as provided by Scikit learn [100].

Table 4.2: Kernel functions [100]

Kernel	Function
RBF	$K(x,x') = \exp(-\gamma\|x-x'\|^2)$
Linear	$K(x,x') = \langle x,x'\rangle$
Polynomial	$(\gamma\langle x,x'\rangle + r)^d = \langle x,x'\rangle$
Sigmoid	$\tanh(\gamma\langle x,x'\rangle + r)$

The decision function, as conceptually introduced in Section 2.5, can be specified according to the kernel chosen for the model [101]. The decision function used for SVM is defined by Equation 4.11

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{n} y_i\alpha_i K(x_i,x) + b\right), \tag{4.11}$$

where sgn() is the sign function, x is the test data, x_i are support vectors, $\alpha_i y_i$ are the support vector coefficients, $K(x_i,x)$ is the kernel function and b is the bias [92].

4.4.2 Hyperparameters

In addition to choosing a kernel, hyperparameters must be tuned to achieve optimal performance. There are many hyperparameters for SVM, but the ones of interest in this work are the regularization parameter C and the kernel coefficient γ. The hyperparameters that are used for the different models in this work varies depending on the kernel chosen. Table 4.3 shows the dependence between kernel and hyperparameters.

Table 4.3: Kernel and hyperparameters

Kernel	hyperparameters
RBF	C,γ
Linear	C

The regularization parameter C provides a trade-off between a smooth decision boundary (hyperplane) and correctly classifying the training datapoints [102]. High values of C will make the model try to classify as many training datapoints as possible correctly, making more complex decision boundaries. Low values of C will make a more general hyper plane. Figure 4.15 illustrates how the decision boundary is affected by a varying C [102].

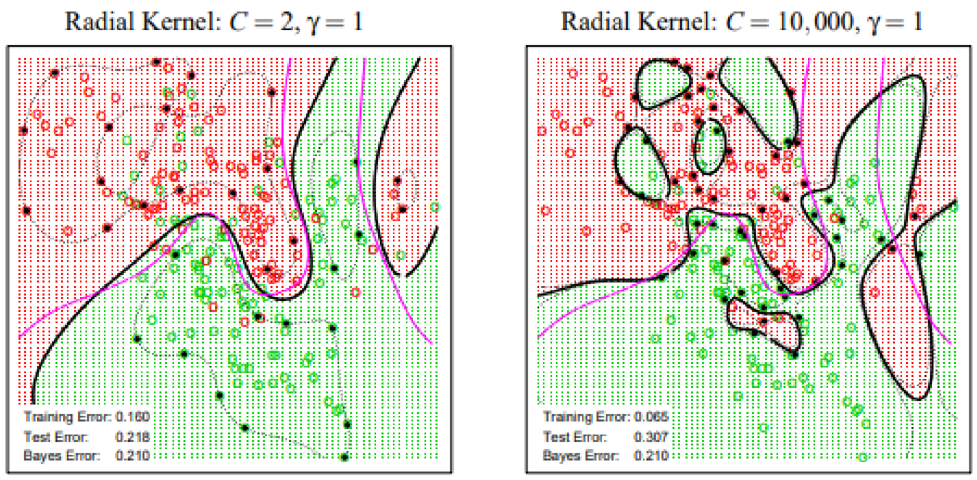


Figure 4.15: Illustration of how the decision boundary is affected by a varying C [102].

γ is called the kernel scale parameter [102] and is used together with C in a SVM based model with RBF kernel. The combination of the two hyperparameters has a high impact on the model performance. A lower γ typically should be paired with a higher C (less regularized decision boundary), and a higher γ should typically be paired with a lower C (more regularized decision boundary). Figure 4.16 illustrates how an SVM classifier with RBF kernel differs in error rate on a test performed with different combinations of γ and C [102].

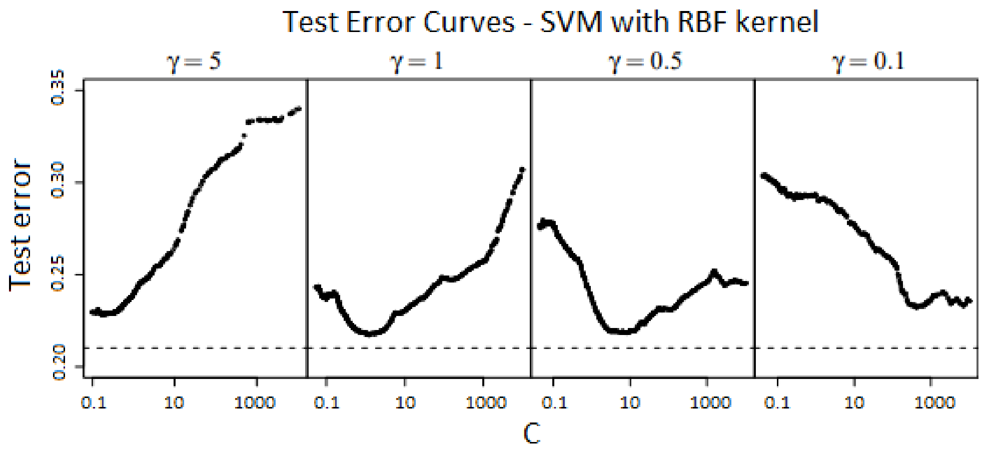


Figure 4.16: Illustration of how γ and C affects the error rates of a RBF kernel-based SVM classifier [102]. The x-axis is log scaled.

Figure 4.16 does not represent what the parameters C and γ should be set to in this work but illustrates that a tuning of the two parameters is important to get good performance.

To find the optimal hyperparameters and kernel for the SVM a grid search provided by Scikit learn, using Cross-validation on the training data, is utilized in this work. A parameter-grid is defined with a list of varying values for the hyperparameters C and γ. A model is trained on the training data for every possible combination of the hyperparameters, and the Cross-validation results are used to decide what model performed the best. This is an extensive search but makes sure that every combination is tested. The values for the hyperparameters that was chosen to be tested in this work is shown in Table 4.4.

Table 4.4: Hyperparameter values for grid search

C	γ	Kernel
10^{-3}	10^{-4}	RBF
10^{-2}	10^{-3}	Linear
10^{-1}	10^{-2}	
1	10^{-1}	
10^{1}	1	
10^{2}		
10^{3}		

The output of a grid search, when applied to the parameter values for kernel, C and γ as in Table 4.4, can be found in Appendix J.

4.4.3 Other classifiers

As mentioned in Section 2.5, there are many machine learning algorithms that works in its own way to solve classification tasks. Section 4.4.1 presented the machine learning algorithm of focus in this thesis, Support Vector Machine (SVM), while this section will give a short introduction of a few other machine learning algorithms mentioned. This section only shows an interpretation of how the algorithms work and is not based on anything found as part of this work. Some examples are however created to illustrate the concepts of some of the algorithms, as part of this work.

Random Forests

The Random Forest classifier is a supervised classifier based on Decision Trees used in machine learning. When training a Decision Tree, it tries to choose the best features and feature values to create "decision rules", used to classify datapoints [103]. The best feature is used at the *root* of the tree, which is the first node in the Decision Tree. Figure 4.17 shows an example of a very simplified Decision Tree classifier, created in this work, that could be used to classify new iris flowers in the iris flower example in Section 2.5.

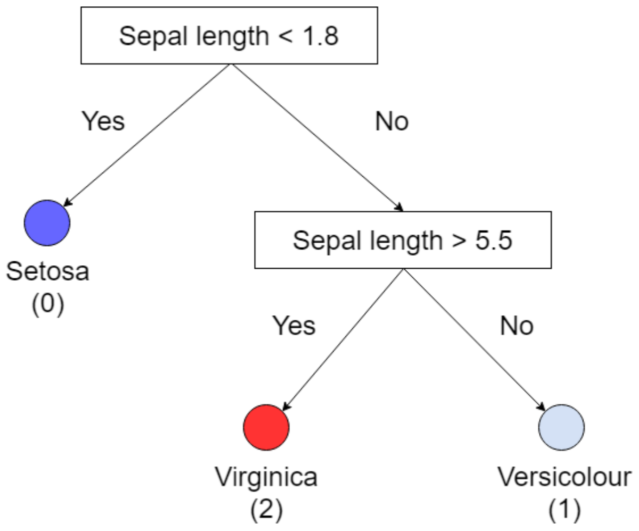


Figure 4.17: Showing an illustration of a simplified Decision Tree classifier, connected to the example shown in Section 2.5.2.

Random Forest is an ensemble method, which means that it combines several classifiers of the same or different type. Then it uses a voting scheme for all the classifiers to output predictions [104]. Random Forest uses several Decision Trees, trained on different subsets of the dataset, and uses a voting mechanism to output the final prediction.

K-Nearest Neighbors

K-Nearest Neighbors (KNN) is another supervised learning classifier and is often used when there is little or no knowledge about the distribution of datapoints [105]. Instead of realizing some decision function or decision boundary through training, the algorithm "saves" datapoints introduced in the training stage. When introducing the classifier to new data points after training, a Euclidean distance is calculated from the new data point to the K nearest datapoints introduced in the training stage. The prediction from the classifier is set to the most frequent true class of the K nearest training datapoints [105]. The variable K can be tuned and impacts the prediction of the classifier. Figure 4.18 shows an illustration of how a new datapoint is classified with a KNN classifier [105].

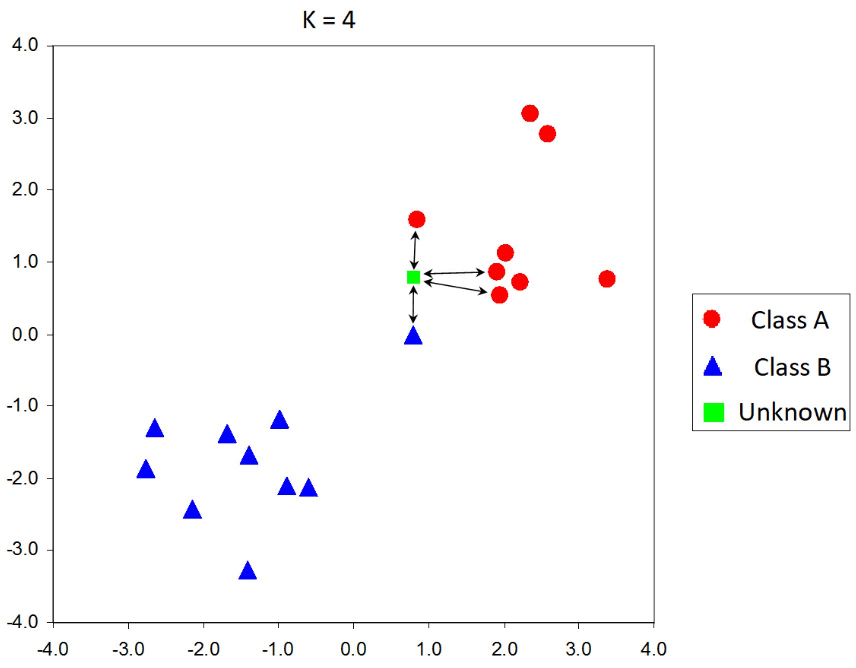


Figure 4.18: Showing an illustration of a how a KNN classifier predicts a new data point [105].

The illustration in Figure 4.18 shows that the number of neighbors to consider is $K = 4$. The new datapoint will be predicted to be the most frequent true class of the $K = 4$ closest datapoints introduced during the training. The most frequent of the $K = 4$ nearest neighbors is class A, which means that the new datapoint would be predicted as class A.

4.5 Machine learning summary

This chapter introduced methods for abstracting the data through feature extraction, to select feature subsets, scale the datasets and split it into data for training and testing and showed that there are a lot of possibilities when choosing a machine learning algorithm. When training a machine learning algorithm, it is important to pay attention to how the models can be *overfitted* and *underfitted*. Having good procedures for testing the models, such as use of Cross-validation and a separate test set, was used in this work to help avoid cases where the model is too specific or generalized towards some data. There is no algorithm that is the obvious choice when solving a classification problem, and the method used to find the best classifier is to experiment with different algorithms and hyperparameters. However, using the theory and techniques presented in this chapter a lot of information can be extracted from the experimental procedure, shortening the process.

Chapter 5

Offline Prediction Results

In this chapter, results regarding machine learning implementations are presented. Several models have been built from several algorithms to give a comparison basis for the proposed solution. Results will be presented as both offline and online results in this work. Offline results are presented in this chapter and are based on a training and test set stored on the computer (online results are based on a scheme where the model is tested on data captured in real time). Two test subjects' data was used to train and test several classifiers. A walk through of the different approaches and results for Subject 1 is presented, and a comparison of classifier performance between the two subjects is given in Section 5.5 as part of a summary. Online results will be presented in Section 6.5, as this is related to control of the drone and cannot be tested with a stored dataset.

All the experiments are first presented with data from Subject 1 and will in Section 5.5 be compared with results from Subject 2. Both subjects have used the same test procedure. Each subject has its own dataset with 1560 chosen datapoints, 260 for each class. The features presented in Section 4.1.2 are now given an index shown in Table 5.1. Only the feature index will be referenced when presenting the features selected for the different models. The dynamic centering of data, as introduced in Section 3.3.6, is only used in the "samples as features" implementation in Section 5.1.1. This is because the classifier was observed to perform better without centered data when extracting features. All sections presenting a new classifier will start with a short introduction specific for the classifier, before presenting the results. Table 5.1 shows the possible features the RFECV algorithm can select from. RFECV was introduced in Section 4.1.4.

Table 5.1: Feature indices, names and channels

Index	Name	Channel
0	Higuchi Fractal Dimension	1
1	Higuchi Fractal Dimension	4
2	Minimum Value Difference	1 and 3
3	Maximum Value Difference	1 and 3
4	Spectral Entropy	1
5	Spectral Entropy	4
6	Pearson Coefficient	3 and 4
7	Pearson Coefficient	3 and 4
8	Pearson Coefficient	1 and 4
9	Pearson Coefficient	1 and 4
10	Covariance	3 and 4
11	Covariance	3 and 4
12	Covariance	1 and 4
13	Covariance	1 and 4
14	Standard Deviation	1
15	Standard Deviation	4
16	Slope	1
17	Slope	4
18	$\theta\beta$ ratio	1
19	$\theta\beta$ ratio	4
20	Petrosian Fractal Dimension	1
21	Petrosian Fractal Dimension	4
22	Peak-to-Peak	1
23	Peak-to-Peak	4
24	Minimum Value	1
25	Maximum Value	1

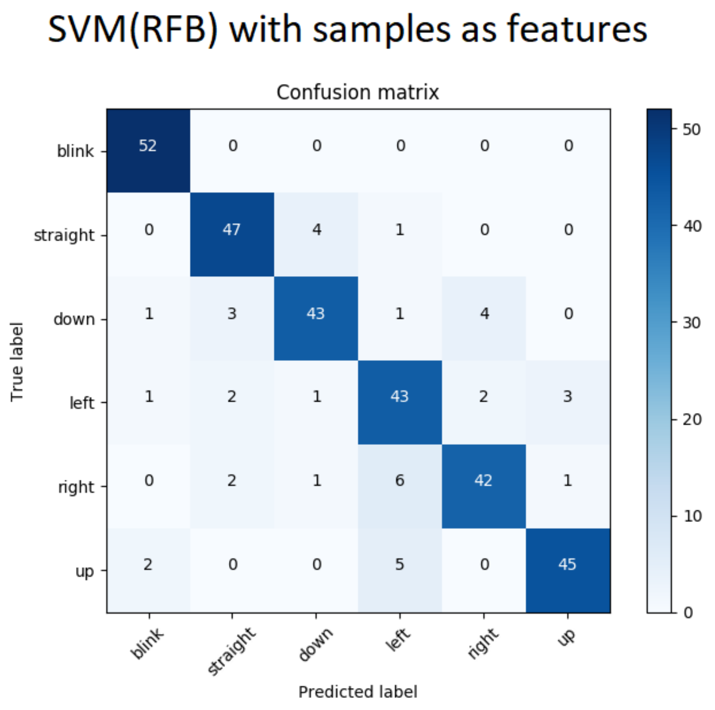
5.1 SVM with RBF kernel

The first classifier to be tested is the SVM with RBF kernel. It will first be tested without feature extraction, where time-series data will be used as input, to determine if feature extraction is necessary. Then it will be tested with features chosen by the RFECV algorithm, as introduced in Section 4.1.4. Cross validation was performed on the training set with $k = 50$ folds for each classifier.

5.1.1 Samples as features

The first experiment was done without extracting features, and instead put pre-processed time-series data $x_m[n]$ into the classifier. It was tested with a number of different channel combinations and window lengths. The best result was obtained with a window length of $l_{w,c} = 50$ from channels $x_1[n]$, $x_2[n]$ and $x_4[n]$. This results in a feature vector of length 150, with 50 features from each channel. Hyperparameters were optimized with the grid-search to $C = 50$ and $\gamma = 0.01$ The Cross-validation result for Subject 1 was an accuracy of $87 \pm 13\%$.

To validate this performance, the trained model is used to predict on a test set. Figure 5.1 shows the Confusion Matrix listing the predictions made on the test set.



	blink	straight	down	left	right	up
blink	52	0	0	0	0	0
straight	0	47	4	1	0	0
down	1	3	43	1	4	0
left	1	2	1	43	2	3
right	0	2	1	6	42	1
up	2	0	0	5	0	45

Figure 5.1: Confusion Matrix for SVM with samples as features on the test set. An ideal model would have 52 true positives for each class.

The accuracy for Subject 1 on the test set was shown to be 87.1%, where the Classification Report in Table 5.2 shows the metrics for the respective classes.

Table 5.2: Classification Report

Class	Precision	Recall	f1-score	support
Blink	0.93	1.00	0.96	52
Down	0.88	0.83	0.85	52
Left	0.77	0.83	0.80	52
Straight	0.87	0.90	0.89	52
Right	0.88	0.81	0.84	52
Up	0.92	0.87	0.89	52
avg/total	0.87	0.87	0.87	312

The reason why this model performs so poorly may be affected by several reasons. The vectors are shortened down to 50 samples for each channel, leaving out vital information. However, increasing it did not result in better performance. This is thought to be because of the "curse of dimensionality" [37], and lack of training data.

5.1.2 Feature extraction

In this experiment, feature extraction was performed on the signal $x[n]$. The features were selected from the proposed features by the RFECV algorithm. It must be noted that the RFECV algorithm does not work on SVM with RBF kernel. Therefore, the optimization with regards to the choice of features were done on a model with identical hyperparameters, with exception of the kernel which is set as linear. The selected features were index: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 22, 23, 24, 25. Giving a total of 23 features.

Cross-validation was performed on the training set with $k = 50$ folds and the hyperparameters were optimized to $C = 10$ and $\gamma = 0.01$. The Cross-validation accuracy on the training set for Subject 1 was shown to be $96 \pm 7\%$. As a reminder, the output from Cross-validation is the average accuracy for all k-folds, $\pm$ the standard deviation. So $96 \pm 7\%$ does not mean that the best result was 103%, as you cannot get more right predictions than you are trying to predict.

To validate this performance, the trained model is used to predict on the test set. Figure 5.2 shows the Confusion Matrix from the validation on the test set.

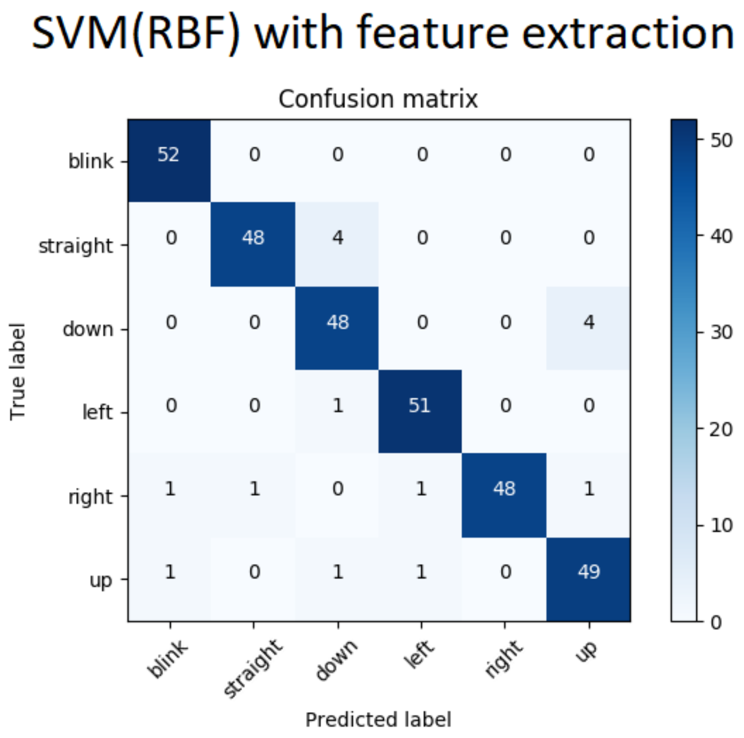


Figure 5.2: Confusion Matrix for SVM based model with RBF kernel on the test set with extracted features. An ideal model would have 52 true positives for each class.

The accuracy on the test set was shown to be 94.9%, where the Classification Report in Table 5.3 shows different scores for the respective classes.

Table 5.3: Classification Report

Class	Precision	Recall	f1-score	support
Blink	0.98	1.00	0.98	52
Down	0.89	0.92	0.91	52
Left	0.96	0.98	0.97	52
Straight	0.98	0.92	0.95	52
Right	1.00	0.92	0.96	52
Up	0.91	0.94	0.92	52
avg/total	0.95	0.95	0.95	312

These results show that there is a huge improvement by performing a feature extraction. Feature extraction introduces additional computation on the data, but in turn yields better classifier performance and a lower dimension of the feature vector. The trade-off between additional computation and classifier performance will be further investigated in Chapter 7.

5.2 SVM with linear kernel

The same features as in Section 5.1.2 were used for this model, as the RFECV algorithm did not support feature selection for the RBF based model. The decision function is the same for this model, as defined by Equation 4.11. The hyper parameter C was optimized to $C = 10$ and Cross-validation was performed on the training set with $k = 50$ folds, achieving an accuracy of $95 \pm 9\%$ for Subject 1. This is a small decrease compared to the SVM based model with RBF kernel.

To validate this performance, the trained model is used to predict on the test set. Figure 5.3 shows the Confusion Matrix from the validation on the test set.

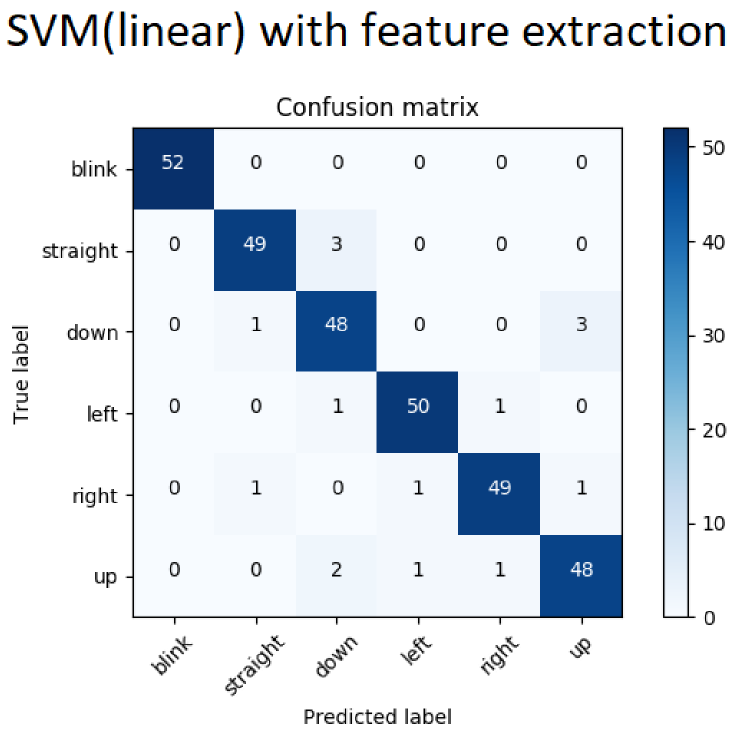


Figure 5.3: Confusion Matrix of the support vector machine with linear kernel. An ideal model would have 52 true positives for each class.

The accuracy on the test set was shown to be $A = 0.949$, where the Classification Report in Table 5.4 shows different scores for the respective classes.

Table 5.4: Classification Report

Class	Precision	Recall	f1-score	support
Blink	1.00	1.00	1.00	52
Down	0.89	0.92	0.91	52
Left	0.96	0.96	0.96	52
Straight	0.96	0.94	0.95	52
Right	0.96	0.94	0.95	52
Up	0.92	0.92	0.92	52
avg/total	0.95	0.95	0.95	312

The Classification Report in Figure 5.4 showed very similar scores for the different classes, as with the SVM based model with RBF kernel.

5.3 Linear Support Vector Classifier

Scikit-learn provides another implementation of SVMs based on another C-library called *liblinear*, while the SVM models presented so far are based on a C-library called libsvm. Linear Support Vector Classifier (LinearSVC) works in a similar matter as the linear kernel SVM but differs in the fact that it inherits from another C-library specifically designed for linear SVMs. It also differs slightly in the way it learns from the datapoints given through training. The decision function in LinearSVC is the same as defined in Equation 4.11

As with SVM, features are extracted from the signal $x[n]$. The features were selected from the proposed features by the RFECV algorithm. All features were selected, giving a total of 26 features. Using Cross-validation with $k = 50$ folds on the training set for Subject 1, an accuracy of $95 \pm 8\%$ was achieved. Which is the same as SVM with linear kernel. The hyperparameters were optimized through a grid search to $C = 10$. To validate this performance, the trained model is used to predict on the test set. Figure 5.4 shows the Confusion Matrix when using the Linear SVC classifier on the test set.

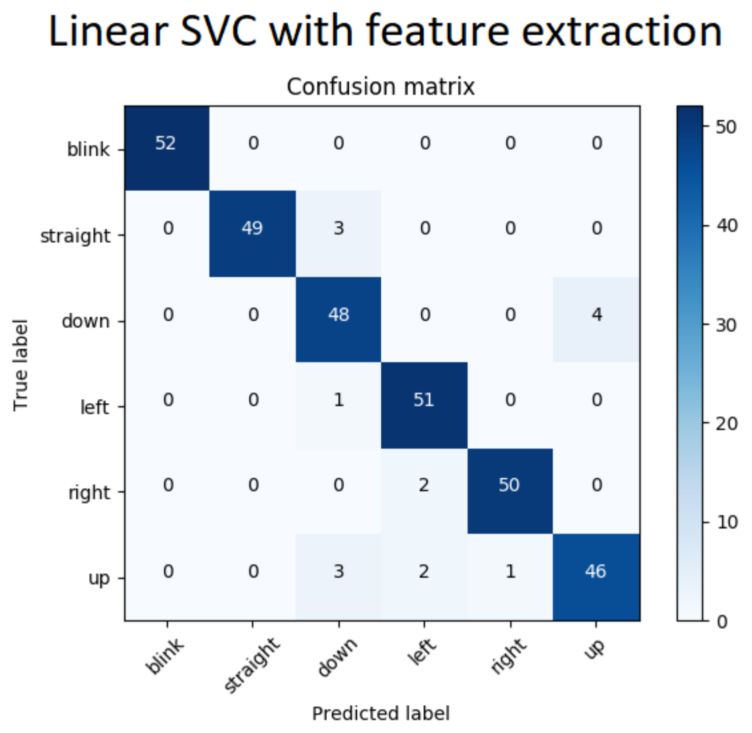


Figure 5.4: Showing the Confusion Matrix of the LinearSVC model. An ideal model would have 52 true positives for each class.

The accuracy on the test set was shown to be 95%, where the Classification Report in Table 5.5 shows different scores for the respective classes.

Table 5.5: Classification Report

Class	Precision	Recall	f1-score	support
Blink	1.00	1.00	1.00	52
Down	0.87	0.92	0.90	52
Left	0.93	0.98	0.95	52
Straight	1.00	0.94	0.97	52
Right	0.98	0.96	0.97	52
Up	0.92	0.88	0.90	52
avg/total	0.95	0.95	0.95	312

The scores presented by the Classification Report in Figure 5.5 are similar to those shown in the other models. The only noticeable change is the Up class, which showed a recall of 88%, whereas the SVM based models showed recall above 92% for all classes.

5.4 Other classifiers

To give comparison basis for the SVM classifiers presented, some other algorithms were also tested. These will not be presented as detailed because they are only used as a comparison basis in this thesis. In this section only the most valuable metrics for Subject 1 will be presented, the rest can be found in Appendix E.

5.4.1 Random forest

Hyperparameters were tuned to:

- number of estimators = 54
- max depth = 30
- min samples leaf = 1

Chosen features were: index: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 22, 23, 24, 25. Giving a total of 23 features. With a Cross-validation with $k = 50$ folds, the classifier achieved an accuracy of $94 \pm 10\%$.

5.4.2 K-nearest neighbors

Hyperparameters were tuned to:

- number of neighbors = 5.

The RFECV algorithm does not support feature selection for KNN due to how KNN works, as introduced in Section 4.4.3. The same features as used for the Random Forest classifier were therefore chosen. With a Cross-validation with $k = 50$ folds, the classifier achieved an accuracy of $94 \pm 9\%$.

5.5 Summary and comparison

A short summary between the different models are presented in Table 5.6 to give a short overview.

Table 5.6: Result summary, Subject 1

Model	Accuracy (CV)	Features (no.)
SVM (RBF), samples	$87 \pm 13\%$	150
SVM (RBF)	$96 \pm 7\%$	23
SVM (Linear)	$95 \pm 9\%$	23
LinearSVC	$95 \pm 8\%$	26
Random forest	$94 \pm 10\%$	23
KNN	$94 \pm 9\%$	23

The presented results so far have been for Subject 1, where this subjects data has been used both to train and test the models. The same features and hyperparameters as found for subject 1 has been used with the data from Subject 2. The reason for not selecting features and tuning parameters for Subject 2 is to see the viability of using one subject to find features and hyperparameters, and another to predict his eye movements by only introducing another dataset. All Confusion Matrices for Subject 2 can be found in Appendix D. Table 5.7 shows a short summary of the accuracies achieved with the different models for Subject 2.

Table 5.7: Result summary, Subject 2

Model	Accuracy (CV)	Features (no.)
SVM (RBF), samples	$81 \pm 17\%$	150
SVM (RBF)	$94 \pm 8\%$	23
SVM (Linear)	$93 \pm 9\%$	23
LinearSVC	$94 \pm 8\%$	26
Random forest	$94 \pm 9\%$	23
KNN	$92 \pm 12\%$	23

It can be seen in Table 5.7 that subject 2 performs well using the same features and hyperparameters as Subject 1. The accuracies for Subject 2 are slightly lower than for Subject 1, where the top performing models achieved an accuracy of $94 \pm 8\%$.

In addition to training and testing on data from the respective subjects, the cross-subject portability was tested by training on one subject and testing on the other. As a final comparison, data from both subjects was used to train a model and tested on the subjects respectively. This is done by appending the training data from the subjects together and use the same test sets as before when testing. The exact same data as with the previous models is used for testing and training, the only difference is that the model trains on data from both subjects, doubling the number of datapoints used for training. The classifier used is the best performing model for Subject 1, SVM with RBF kernel, with the hyperparameters $C = 10$ and $\gamma = 0.01$. A Cross-validation with $k = 50$ folds for the model trained on both subjects achieved an accuracy of 94 $\pm$7%. Table 5.8 shows a subject comparison of the accuracies achieved.

Table 5.8: Subject Comparison. Each cell is the accuracy of the model when trained on a subject (rows) and tested on another subject (columns).

Subject	1	2	Both
1	94.9%	76.2%	N/A
2	76.6%	94.2%	N/A
Both	93.6%	92.6%	93.1%

The Confusion Matrices for the test sets from the different subject combinations can be found in Appendix F. As a reminder, the results are only presented from one of the many possible splits of training and testing. Both worse and better test results can be found when altering this split.

It can be seen in Table 5.8 that the results from testing on a subject is dependent on the subject(s) used for training the model. Training on Subject 1 and testing on Subject 2 showed an 18% decrease in accuracy from 94.2% to 76.2% compared to training and testing on Subject 2. Using the test data from Subject 1 when training on Subject 2 showed an 18.3% decrease from 94.9% to 76.6% compared to training and testing on Subject 1. Both subjects showed a small decrease in accuracy when training a model on both subjects but lies around the accuracy that was shown when training and testing on them, respectively. Both subjects also showed accuracies within the range given from the Cross-validation result of 94 $\pm$ 7%, when training a model on both subjects.

It can be concluded that the classification is poor when training on one subject and testing on another. However, a single model can provide good results for both subjects by introducing data from both subjects during training.

All results presented in this chapter were with a window length of $l_{w,c} = 250$. A window length of 250 equals a time period of 1 second as the sampling frequency is $f_s = 250$ Hz. The models presented are specific to their window lengths, varying the window length will therefore result in a different model. This is something to keep in mind when reading further. The window length of 250 was set as a basis and showed great results offline, this is not necessarily the case for online classification. A proof of concept was shown in this chapter, using offline results as a measure of its success. The two top performing classifiers from this chapter, SVM based model with RBF kernel and LinearSVC, will be implemented in an online classification scheme to see if the same classifiers can be used in a real-world classification application.

Chapter 6

Controller and Online Results

To use predictions from the classifier to issue commands to a drone, a controller is needed. The drone used is a Parrot AR.drone 2.0 [106]. It does not come with a Python API from the manufacturer, but there are 3rd party APIs available. The API used for this project is the ps_drone API by J. Philipp de Graaff [47]. The controller-logic will be used together with the predictions to give a measure of online classification performance. The online classification performance can therefore also give an indication of how accurately the drone can be controlled using the predictions as control inputs. Figure 6.1 gives a quick recap of signal names and modules in the system closely related to the controller.

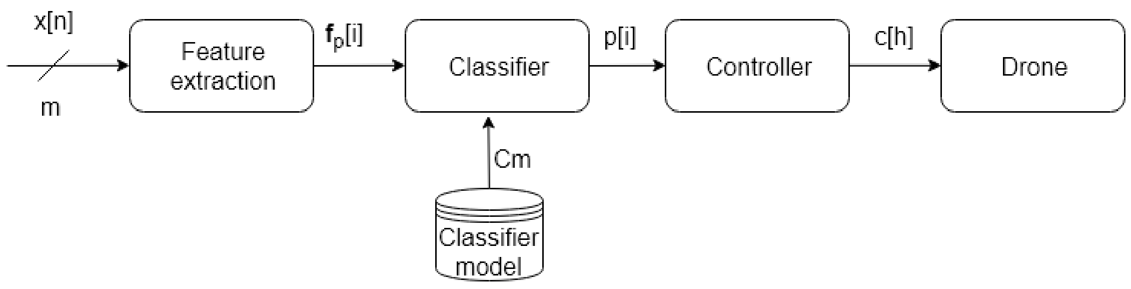


Figure 6.1: Block diagram with an overview of essential signal names.

It can be seen in Figure 6.1 that the signals $p[i]$ and $c[h]$ have different indexing. This is because they are produced at a different rate. The predictions $p[i]$ are produced at a fixed rate, and the commands to the drone $c[h]$ are issued at irregular intervals, depending on the sequence of $p[i]$. This chapter presents the design of the controller and the online experiments performed in this work.

6.1 Online feature extraction and classification

When predicting, features are calculated from the most recent pre-processed samples $\mathbf{x}[n]$ over a window of length $l_{w,c}$, for each prediction. The prediction frequency is found by Equation 6.1 with $o = 75\%$ overlap. The $o = 75\%$ overlap is chosen both from experimental tuning and because T. Hörmann et al. [107] found this to be optimal for SVM classifiers.

$$f_p = \frac{f_s}{(1-o)\, l_{w,c}} = 5 \text{ Hz}, \tag{6.1}$$

where:

- $f_s = 250$ Hz is the sampling frequency
- $l_{w,c} = 200$ is the window length. Will be further explained in Section 6.3

In energy efficiency terms it is favorable to have a low prediction rate, as this means fewer predictions (less calculation). However, any overlap less than $o = 75\%$ was shown in [107] to degrade the performance of the classifier, and any higher did not provide any significant gain in classification accuracy. The rate of predictions is therefore kept to be 5 Hz to keep the overlap of $o = 75\%$.

6.2 Controller design

There are some alternatives to how one can design the controller. An observation of the predictions shows that there are some glitches (wrong predictions). Dealing with these wrong predictions was the main concern when designing the controller. There are ways to deal with this, for example one could design a state-machine with some kind of glitch tolerance. We can look at a series of predictions in Equation 6.2 during an "Up" movement, to get a pinpoint on how to do this. The earliest arriving prediction is the one to the left and the most recent is the one to the right.

$$p[i] = \ldots, 5, 5, 5, 8, 8, 0, 8, 2, 5, 2, 2, 2, 2, 2, 5, 5, 5, 5, \ldots \tag{6.2}$$

By looking at the predictions in Equation 6.2, we can see that two consecutive predictions can be used to trigger a directional command to the drone. A mechanism is also needed to reliably return to the initial state when the predictions settles on a series of $p[i] = 5$ predictions. Another observation from Equation 6.2 is that the first prediction after a sequence of $p[i] = 5$ predictions is often correct. These observations are used as a basis for the designed state-machines, as will be introduced shortly.

As a reference, the translation of predictions to drone commands can be seen in Table 6.1

Table 6.1: Drone commands based on out-coming predictions

Label $p[i]$	Eye movement	Drone movement $c[h]$
0	< 3 Blinks	None
0	3 Blinks in 3s	Takeoff or land
2	Down	Fly backwards
4	Left	Rotate counter clockwise
5	Returned to center	Hover
5	Look straight	None
6	Right	Rotate clockwise
8	Up	Fly forwards

Before introducing the state-machine we first need to introduce the "Other" and "Opposite" translation dictionaries, defined in Tables 6.2a and 6.2b.

Table 6.2: Translation dictionaries

(a) "Opposite"

Input	Output
2	8
4	6
6	4
8	2
0	0

(b) "Other"

Input	Output
2	4,6
4	2,8
6	2,8
8	4,6
0	2,4,6,8

The purpose of these dictionaries is to identify how the consecutive predictions $p[i]$ relate to the movement being performed $c[h]$. So, let's say for example the drone is flying forwards after a series of "eye up" predictions $p[i] = 8$, then the opposite prediction would be "eye down" $p[i] = 2$. The "Opposite" dictionary shows all the different predictions and their opposites. The

"Other" dictionary is for identifying the predictions that is not the opposite, but rather any other directional prediction. Both these dictionaries are used to evaluate when the drone should stop its current movement and return to an idle state, where it can accept a new movement command $c[h] \in \{2,4,6,8\}$.

Figure 6.2 shows the state-machine diagram for the directional commands. The actual code implementation of the state-machines can be found in Appendix H and are designed as a five-layer deep if-tree. This is not necessarily a good implementation, but it is implemented that way for readability and could easily be flattened to two layers at the expense of more complex statements. The diagrams are produced by best effort to capture the essence of this if-tree, so where it gets complicated, it might be better to look at the implementation in the appendix. Variable names in the diagrams corresponds with the code in the appendix.

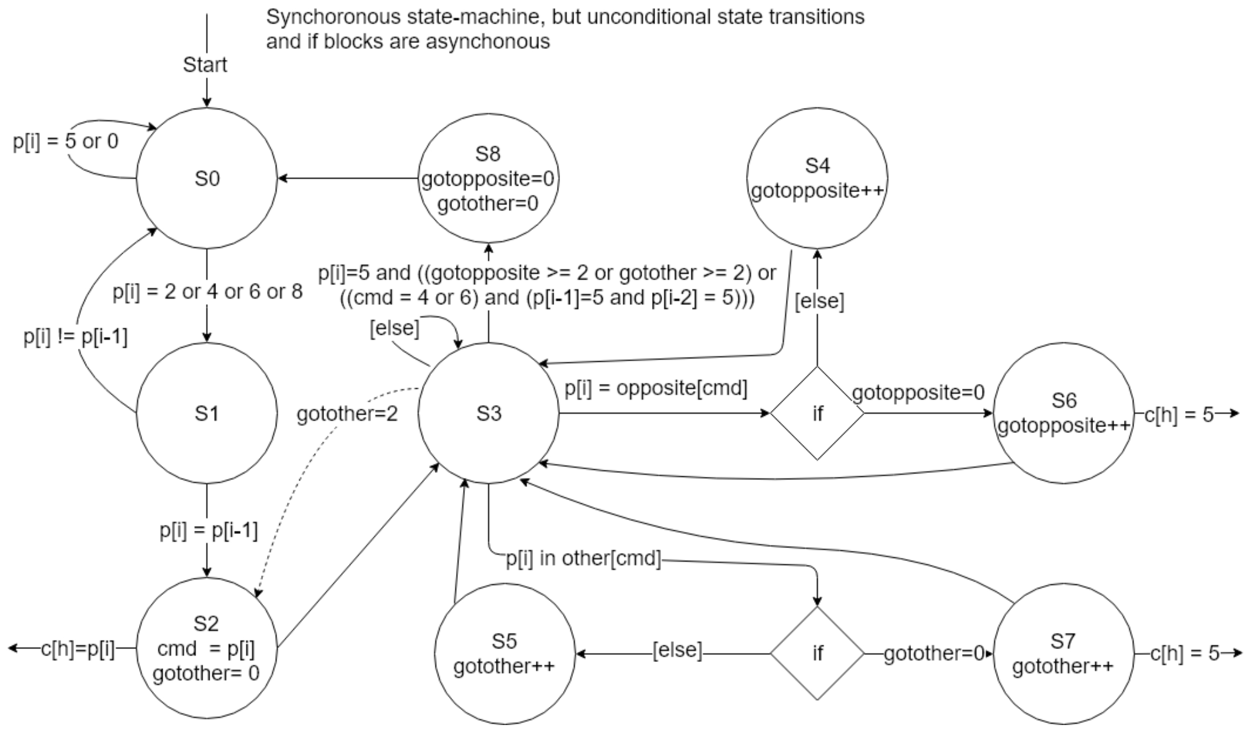


Figure 6.2: State-machine for interpreting the predictions and issue directional commands to the drone.

As Figure 6.2 shows, the glitch tolerance is implemented so that two consecutive predictions of the same class is needed to transmit a directional command to the drone (S2). Any less than two consecutive predictions of the same class will increase the number of undesired drone commands. Increasing the number of consecutive class predictions will cause the controller to miss more of the desired drone commands, refer to Equation 6.2. The rest of the state-machine diagram is a bit complicated, but the purpose is to issue the "hover" command $c[h] = 5$ as fast as possible when the eyes return to center (S7 and S6), and then determine when predictions has settled after the movement and get ready for the next directional command. Although the "Opposite" and "Other" conditions are seemingly redundant, there is a reason for differentiating between "Opposite" and "Other" directions. The reason is the opportunity to add a state transition from S3 to S2 for going directly between two directions (shown in dotted lines). This was tested, but only worked for some of the better classifiers tested online. In order to make a more general-purpose state-machine that works good on classifiers with lower precision, this state transition from S3 to S2 is removed for the online testing. However, if you have a very precise classifier, this is a nice feature to include as the flying experience becomes a bit more intuitive as you do not need to wait for the state-machine to return to S0 before doing a new movement.

Figure 6.2 shows the state-machine diagrams for the takeoff and land commands, controlled by blinks.

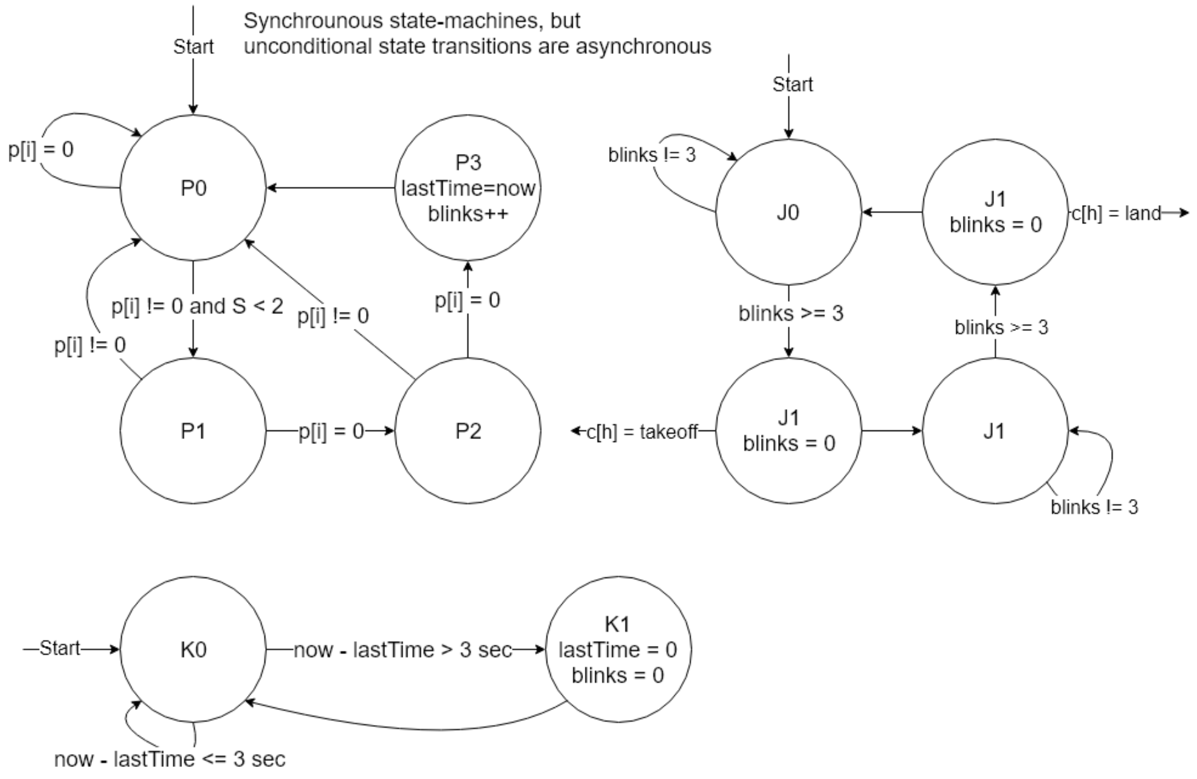


Figure 6.3: State-machines for interpreting the Blink predictions and issue takeoff and land commands to the drone.

In order to determine when to start or land the drone, three state-machines are designed. The first is a blink counter. The second resets the blink counter if more than three seconds has elapsed since the last blink. While the third observes if the drone is flying or not and issues the correct fly/land command if the blink counter reaches 3. The "now" variable is the variable that holds the system time and is updated for each prediction $p[i]$. S is the state of the directional state-machine where $S < 2$ means that the state of the directional state-machine must be either $S0$ or $S1$.

As a feedback to the user a text-to-speech engine is utilized, keeping the user updated with the commands sent to the drone. When running the system on Windows the pywin32 library [108] is used, and for Linux the eSpeak [109] is used. The reason for using different Operating Systems (OS's) is that the PS-drone API needs a posix [110] compliant OS and when not flying the drone, a Windows system is used. Every time a command $c[h]$ is sent to the drone, or the transition from S8 to S0 is made, this is announced to the user through speakers or connected headphones. This makes it easier to understand what the drone is "thinking" and control it accordingly.

6.3 Window size and model selection

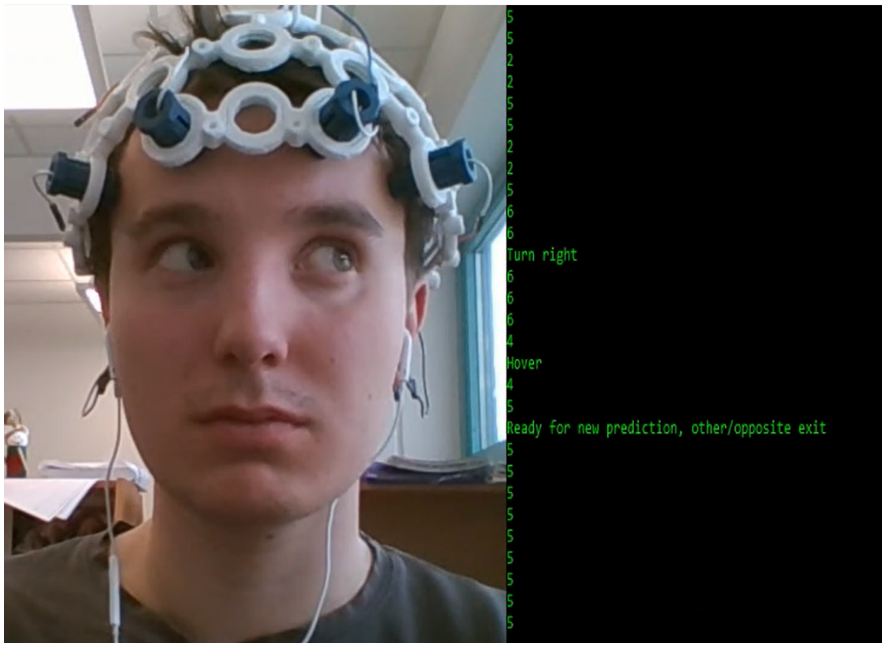
Through initial testing of the controller, it was quickly observed that the window length $l_{w,c}$ highly affects the responsiveness of the system. Having a shorter window length means shorter time periods for each movement which allows for more movements in quick succession. The classifiers presented in the offline results Chapter 5 used a window length of $l_{w,c} = 250$, where the whole window was dedicated to one movement. This means that only movement could be registered every second (sampling frequency $f_s = 250$ Hz). This is very slow when used in practice and makes it difficult to control the drone intuitively as one must be vary to not perform more than one movement within that second.

In addition to the responsiveness, varying the window length $l_{w,c}$ also alters the classifier performance. As many features are calculated over the whole window, the length can affect the importance of the features. The feature value might not have the same importance for the classification when calculating over a different number of samples (window size), as some samples might be left out when reducing the window size. In short, varying the window length $l_{w,c}$ results in a different model which can have different performance in form of accuracy, recall etc. Thus, the best performing classifier from Chapter 5 might not be the best when tweaking the window size for online classification.

An experimental approach was chosen to find the middle ground between performance metrics and responsiveness, by tweaking the window length $l_{w,c}$ and testing these lengths on different models. The starting point was based on the results from Chapter 5 with the best performing classifiers (LinearSVC and SVM with RBF kernel).

6.4 Evaluation method

Evaluating the online classifier was more complicated compared to offline classification, as the test data is the continuous stream of calculated features from the scalp electrodes. The model can still be evaluated on the training data using CV, but a method for evaluating online performance is needed (how CV works was explained in Section 4.3.2). Instead of looking at how the drone moves, a terminal window is used to print the state of the controller and the stream of predictions. Using a web camera to record the eye movements while observing the prints in terminal window, it is possible to count the True Positive, True Negative, False Positive and False Negative. For this, a recording environment is setup using the Open Broadcaster Software (OBS) [111]. This allows recording of both web camera and terminal window at the same time. Figure 6.4 shows the environment used.



```
5
5
2
2
5
5
2
2
5
6
6
Turn right
6
6
6
4
Hover
4
5
Ready for new prediction, other/opposite exit
5
5
5
5
5
5
5
5
5
```

Figure 6.4: Picture of the environment used for online evaluation.

By looking at the recordings after performing a series of movements, the performance of the online classification can be evaluated. The video is recorded at 30 Frames Per Second (FPS). As a little side-note, the system delay can now be calculated by counting the frames from a movement was started till the system issued a command $c[h]$. It was found that 13 frames elapsed before the command was issued. This gives the system a total delay of 430 ms. This delay is mainly due to prediction frequency and overlap between windows. The predictions are made in $1/f_p = 200$ ms intervals, so about half the delay can be accounted for from that. The other half is a bit vague, but it is not unrealistic that around 50 samples from a movement must be within the sliding window before the classifier predicts that movement. If this is the case, around 400 ms of delay is accounted for. The EEG headset itself is another source of delay, but this is not investigated as the delay is considered small enough to not be an issue. It takes 296 μs to perform a prediction (will be shown in Section 7.4), so prediction time is negligible in this context. The delay did, however, not seem to be an important factor when controlling the drone.

The movements are performed in an undefined order, but the number of each movement should be approximately the same (looking straight is an exception). Scores for accuracy, precision and recall are calculated as an unweighted average for all classes. The precision, recall and f1-scores found in the Classification Reports are calculated for the respective classes where the bottom line of the Classification Report shows the weighted average score across all classes. Confidence intervals for the accuracy are calculated with a Wilson interval [112], with a confidence level of 95%. The formula can be seen in Equation 6.3 [112].

$$\frac{\hat{p}+\frac{z^2}{2n}}{1+\frac{z^2}{n}} \pm \frac{z}{1+\frac{z^2}{n}}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}+\frac{z^2}{4n^2}}, \tag{6.3}$$

where:

- $\hat{p}$ is the binomially-distributed observation
- n is the number of observations
- $z = 1.96$

The scores for online accuracy differ from the offline accuracies presented in Chapter 5. The offline accuracies in Chapter 5 was presented as the result of Cross-validation on the training data, ± the standard deviation. The accuracies for online classification is given with a confidence interval instead of the standard deviation. This is because the offline results were calculated using Cross-validation on the training data, while online results show scores from online tests.

When evaluating performance, a True Positive (ref. 4.3.1) for the different directional eye movements and blinks was counted if the drone command $c[h]$ matches the eye movement performed by the subject. A True Positive for the Straight class was counted if no directional drone command $c[h] = 5$ is made while the subject looks straight ahead. Meaning that if no directional drone command was transmitted before an actual movement was performed by a subject, a True Positive for the Straight class was counted. A wrong drone command $c[h]$ (ex. looked right and command to left $c[h] = 4$) will count as False Negative for the direction the drone should have flown in, and a False Positive for the one it actually flew to.

Straight and Blink are considered to be the most important classes, because the drone should stay still when looking straight ahead and the Blink class should be able to reliably control take-off and landing intentionally. In an attempt to increase the correctness of the Straight and Blink predictions, a "no-event" class multiplier m_{nc} was introduced. This is a multiplier that increases the number of datapoints used when training the classifier, for the Straight and Blink classes.

This results in an unbalanced dataset, where the Straight and Blink classes has a higher representation. Training on too many Straight and Blink datapoints will however degrade the performance of the other classes, so it is important to tune this multiplier.

As the scores are determined by looking at a video recording of the test subject and terminal output, there will be some kind of subjective evaluation of what is counted as correct. But the general rule used for these experiments is that a period of staring straight (5) before a movement, equals a correct Straight prediction if no directional drone commands were transmitted during that period. When a movement is performed, the prediction $(2,4,6,8)$ is counted as wrong or correct depending on the command sent to the drone. This method has some weaknesses as no True Positive, False Positive or False Negative are counted for stopping the movement (returning to center position). This means that it is possible that even though the subject has looked back to center (from left position), and is currently looking straight, the state of the machine might tell the drone to keep flying left due to a miss-prediction. There is currently no evaluation method for such cases.

6.5 Online Results

With the method for validation and evaluation as proposed through Section 6.4, several online tests were performed on several models based on LinearSVC and SVM with RBF kernel. The models were tested with varying window sizes $l_{w,c}$ and "no-event" multipliers m_{nc}. Training was done in the exact same way as in Chapter 5, with an exception of introducing more datapoints for the Blink and Straight classes through the "no-event" multiplier m_{nc}. The only difference is that the test set now is the continuous stream of feature extracted electrode recordings. Features were selected with the RFECV algorithm (as explained in Section 4.1.4) for each model. By experimenting with different window sizes and "no-event" multipliers, it was found that the best results was obtained by having a window size of $l_{w,c} = 200$, no dynamic centering of the data (as introduced in Section 3.3.6 and a "no-event" multiplier $m_{\text{nc}} = 1.2$. This results in an increase to 1664 datapoints used from the datasets, 260 for Left, Right, Up and Down while Blink and Straight has 312 datapoints each.

This section presents the observed online results with window size $l_{w,c} = 200$ and "no-event"-class multiplier $m_{\text{nc}} = 1.2$. First when training and testing on the respective subjects, then when training on both subjects and testing on them respectively. The features selected by the RFECV algorithm for each model can be found in appendix I. The Confusion Matrices are given in percent because of the unbalanced nature of the tests, each Confusion Matrix should therefore be studied carefully. The number of different movements performed in the tests is given in the figure texts. The classifier hyperparameters are as with offline classification, $C = 10$ and $\gamma = 0.1$.

6.5.1 Testing with all classes

It was observed that the Down class confuses the classifier both when based on LinearSVC and SVM. This degrades the control of the drone drastically. This was the case for both subjects over a series of different models and was quickly concluded as a problem for proper control of the drone. To illustrate the confusion from the Down class, two tests where Subject 1 tests on a SVM based model with RBF kernel and Subject 2 tests on a LinearSVC based model are highlighted.

Subject 1, SVM based model with RBF kernel

Figure 6.5 shows the Confusion Matrix from a test with all movements.

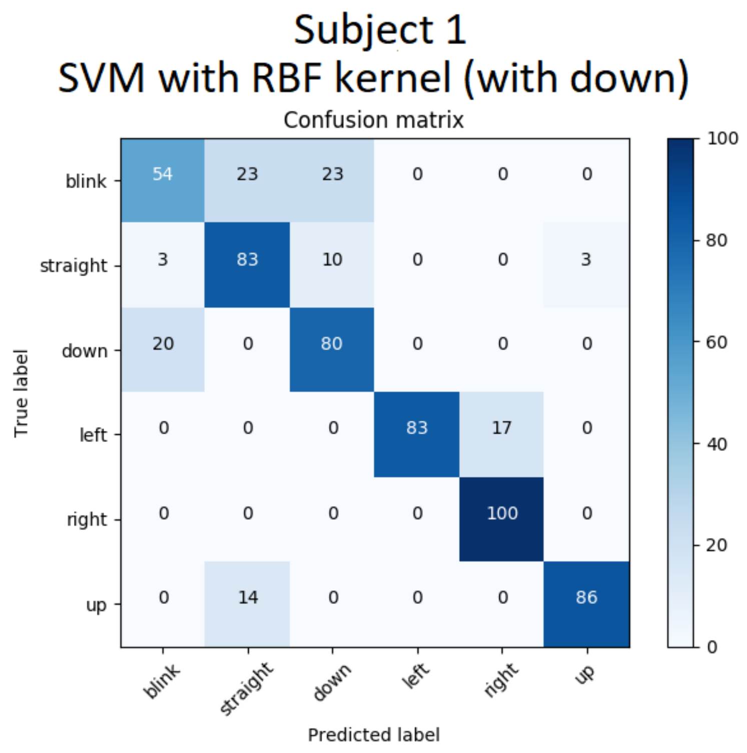


Figure 6.5: Confusion Matrix for SVM with RBF kernel while training on Subject 1 and testing on Subject 1 online with extracted features. An ideal model for this test case would have 13 true positives for Blink, 30 for Straight, 6 for Left, 5 for Right, 5 for Down and 7 for the Up class.

The model achieved an accuracy of 78.8 (67.5, 86.9)% , precision of 78.8% and recall of 81.0%. These are not bad results. It can be seen in Figure 6.5 that the Blink class has about half its predictions correct, while the other half is predicted as Straight and Down. The Classification Report in Table 6.3 shows different scores for the respective classes.

Table 6.3: Classification Report SVM Subject 1

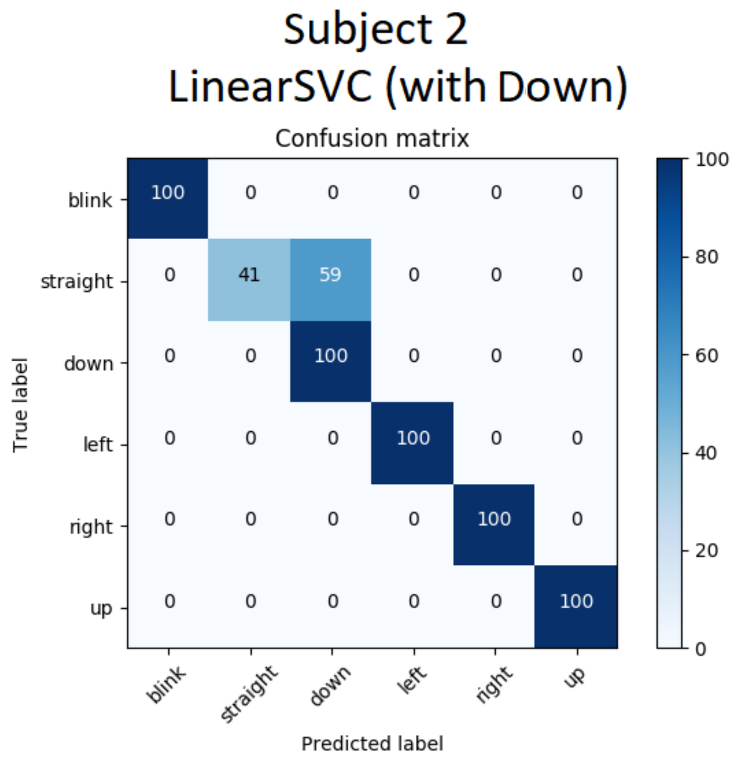
Class	Precision	Recall	f1-score	support
Blink	0.78	0.54	0.64	13
Down	0.40	0.80	0.53	5
Left	1.00	0.83	0.91	6
Straight	0.86	0.83	0.85	30
Right	0.83	1.00	0.91	5
Up	0.86	0.86	0.86	7
avg/total	0.82	0.79	0.93	66

Some scoring metrics to notice are the recall for Blink and precision for Down in Table 6.7.

Blink had a recall of 54% and Down had a precision of 40%, which is very low. The number of eye movements performed in this test is also low but was considered enough because of the poor performance and the illustration purposes.

Subject 2, LinearSVC

The same validation procedure was used for Subject 2 but testing on a LinearSVC based model instead of SVM. Figure 6.6 shows the Confusion Matrix from a test with all movements.



	blink	straight	down	left	right	up
blink	100	0	0	0	0	0
straight	0	41	59	0	0	0
down	0	0	100	0	0	0
left	0	0	0	100	0	0
right	0	0	0	0	100	0
up	0	0	0	0	0	100

Figure 6.6: Confusion Matrix for LinearSVC while testing on Subject 2 online with extracted features. An ideal model for this test case would have 4 true positives for Blink, 51 for Straight, 1 for Down, 7 for Left, 6 for Right and 4 for the Up class.

Figure 6.6 shows the confusion between Straight and Down, resulting in an overall accuracy of 58.9 (47.4, 69.5)%, precision of 83.9% and recall of 90.2%. The predictions for all the classes except Straight are very good on Subject 2. Straight was, however, more often than not predicted as Down. This makes it hard to tell if a Down prediction was from a miss-prediction from the Straight class, or if it was a correct prediction. The Classification Report in Table 6.4 shows different scores for the respective classes.

Table 6.4: Classification Report Linear SVC Subject 2

Class	Precision	Recall	f1-score	support
Blink	1.00	1.00	1.00	4
Down	0.03	1.00	0.06	1
Left	1.00	1.00	1.00	7
Straight	1.00	0.41	0.58	51
Right	1.00	1.00	1.00	6
Up	1.00	1.00	1.00	4
avg/total	0.99	0.59	0.70	73

As with the Confusion Matrix, Table 6.4 shows that the recall of the Straight class and precision of the Down class are very low. As with Subject 1 the number of eye movements are low but the Straight class, that causes the most trouble, has a total number of 51 predictions. Which in this case is considered sufficient as a result of the bad performance and illustration purposes.

Evaluation

Both subjects showed weaknesses when using all four directions in addition to blinking and looking straight ahead. Subject 1 showed a weakness with the confusion for Blink, which was often predicted as Straight and Down, while Subject 2 showed a large confusion between Straight and Down. Straight and Blink are considered important classes as Straight is a "no-event" class, while Blink is used to start flying and landing the drone. No conclusion could be made to as of why the Down class is miss-predicted. An idea to increase performance was to not allow Down to count as a directional command in the state-machine. By discarding the "Down" class as a directional command, it is possible to achieve better control at the cost of having only three directional movements. The only consequence is that the drone will not be able to fly backwards. Instead, the drone needs to rotate 180° before flying forwards to substitute the Down command. The newly proposed state machine is tested in the next sections.

6.5.2 New state-machine without Down

The new state-machine for the directional commands can be seen in Figure 6.7.

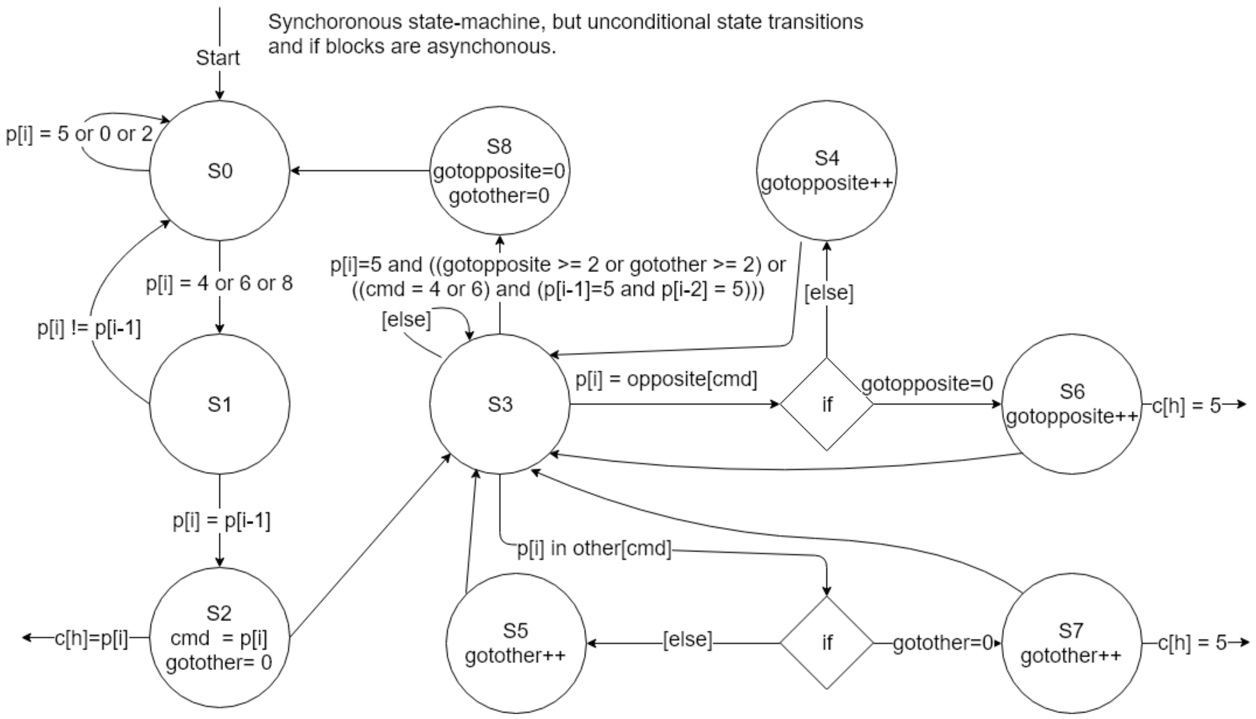


Figure 6.7: The new directional state-machine, without Down as a command.

It can be seen in Figure 6.7 that the new state-machine is similar to the old one, with some minor changes. The state transition between S0 and S1 is changed to only include classes 4, 6 or 8. As a consequence, the loop back now also includes class 2. This is the state-machine that will be used for the rest of the thesis.

6.5.3 Online testing with LinearSVC

Tests were performed on the models created from LinearSVC based on the liblinear C-library [113]. The models are expected to have a better performance than those presented in Section 6.5.1. The Down class is no longer considered as a direction for the drone to fly in, therefore the Confusion Matrices and Classification Reports does not contain the Down class. The Down predictions $p[i] = 2$ is only discarded from the state transition from S0 to S1 in the state machine, otherwise it is used as before.

Subject 1

Figure 6.8 shows the Confusion Matrix when testing Subject 1 on the LinearSVC based model.

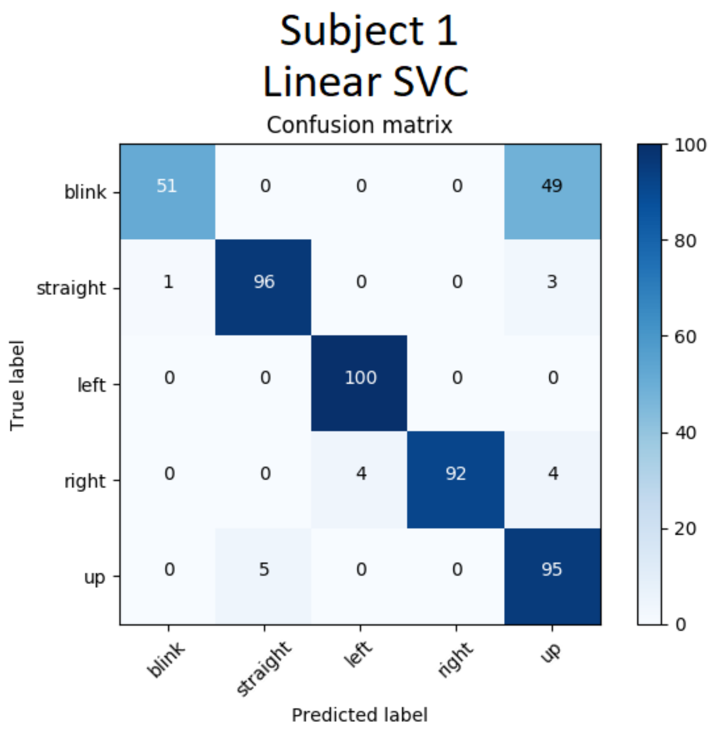


Figure 6.8: Confusion Matrix for LinearSVC while testing on Subject 2 online with extracted features. An ideal model for this test case would have 37 true positives for Blink, 108 for Straight, 23 for Left, 24 for Right and 22 for the Up class.

When comparing Figure 6.8 to Figure 6.5 it can be seen that the predictions has changed slightly for the Left, Right, Up and Straight classes. Something to notice is the fact that Blink still is confused, but this time it miss-predicts as the Up class instead of Straight and Left. This confusion between Blink and Up was not present in the SVM based model in Section 6.5.1. It is possible that this confusion is present due to the fact that the model is based on LinearSVC, which should be investigated. The overall performance of the presented model based on LinearSVC achieved an accuracy of 88.3 (83.3, 92.0)%, precision of 87.7% and recall of 87.0%. This is an increase from the state-machine using all movements as in Section 6.5.1. The Classification Report in Table 6.5 shows different scores for the respective classes.

Table 6.5: Classification Report LinearSVC Subject 1

Class	Precision	Recall	f1-score	support
Blink	0.95	0.51	0.67	37
Left	0.96	1.00	0.98	23
Straight	0.99	0.96	0.98	108
Right	1.00	0.92	0.96	24
Up	0.49	0.95	0.65	22
avg/total	0.93	0.88	0.89	214

The number of eye movements performed in each direction has increased in this test, compared to the test in Section 6.5.1. The Classification Report in Table 6.5 shows that the recall is high for all classes except Blink, which is at 51%. It can also be seen that Blink has a very high precision of 95% which means that it is almost always correct when a Blink is actually predicted. The low recall of the Blink class affects the precision of the Up class, giving it a precision of 49%. Overall, the model performs well on all classes except Blink and Up. Finding a model which does not get confused between these two classes would result in a very good classifier.

Subject 2

Figure 6.9 shows the Confusion Matrix when testing Subject 2 on the LinearSVC based model.

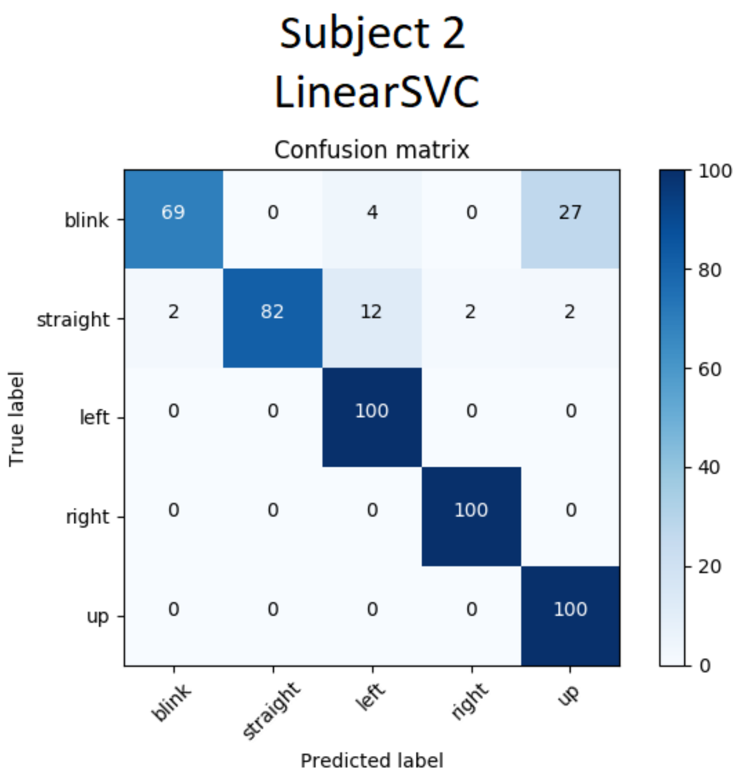


Figure 6.9: Confusion Matrix for LinearSVC while testing on Subject 2 online with extracted features. An ideal model for this test case would have 52 true positives for Blink, 133 for Straight, 24 for Left, 24 for Right and 24 for the Up class.

It can be seen in Figure 6.9 that the Straight class has improved compared to the state machine using the Down class as in Figure 6.6. The Right, Left and Up classes are very accurate with an accuracy of 100%. There are, however, weaknesses shown for the Blink and Straight classes. Blink is as with Subject 1 mistaken as the Up class, but with a smaller degree of confusion than for Subject 1. 12% of the straight-ahead events were also predicted as the Left class. Subject 2 achieved an overall accuracy of 84.4 (79.5, 88.4)%, precision of 80.1% and recall of 90.2% when not using the Down class in the state-machine. This is an increase of 25.5% in accuracy compared with the state-machine including the Down class as a directional command. The Classification Report in Table 6.6 shows different scores for the respective classes.

Table 6.6: Classification Report LinearSVC Subject 2

Class	Precision	Recall	f1-score	support
Blink	0.92	0.69	0.79	52
Left	0.57	1.00	0.73	24
Straight	1.00	0.82	0.90	133
Right	0.92	1.00	0.96	24
Up	0.59	1.00	0.74	24
avg/total	0.90	0.84	0.85	257

As expected from looking at the Confusion Matrix in Figure 6.9 it is seen that the precision of the Left class is low due to the miss-predictions when looking straight ahead. The Up class can be seen to have low precision due to the fact that blinks are frequently miss predicted as Up.

Evaluation

The LinearSVC based models showed some distinct weaknesses for both test subjects by confusing Blinks for the Up class. This makes the control of the drone bad in practice, as the drone could unintentionally start flying forwards. The classifier showed strengths in classifying right, left and up movements but does not make up for the weakness showed for the blinks. It would be impossible to control the drone properly with this classifier, as flying forwards unintentionally clearly is a problem.

6.5.4 Online testing with SVM

A SVM based model with RBF kernel showed the best results in the offline results. It is therefore expected that this model also will perform better than the other classifiers when testing online.

Subject 1

Figure 6.10 shows the Confusion Matrix when testing Subject 1 with a model based on SVM with RBF kernel.

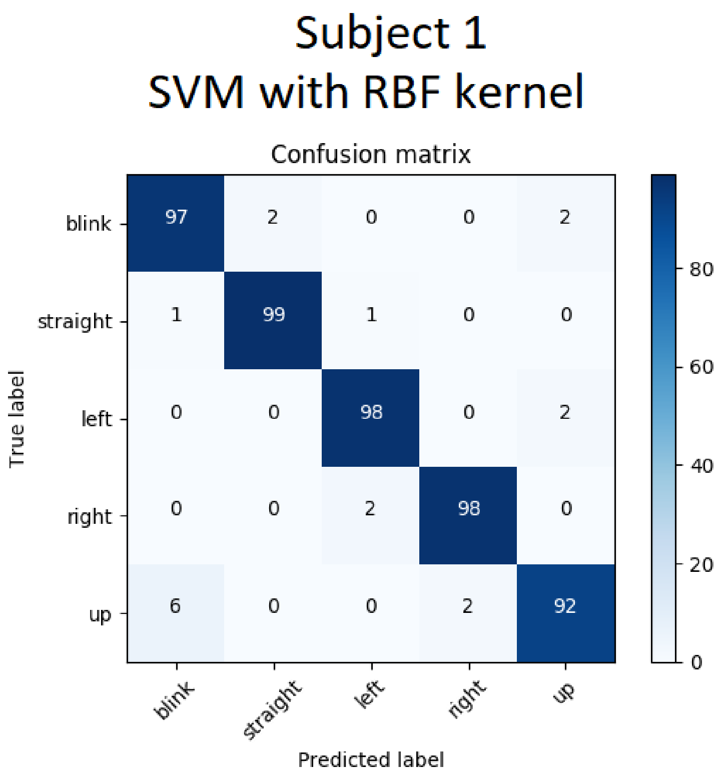


Figure 6.10: Confusion Matrix for SVM with RBF kernel while training on Subject 1 and testing on Subject 1 online with extracted features. An ideal model for this test case would have 58 true positives for Blink, 189 for Straight, 45 for Left, 43 for Right and 49 for the Up class.

Figure 6.10 shows that the model performs very well. All classes are distinguishable by the model, resulting in an accuracy of 97.4 (95.3, 98.6)%, precision of 96.4% and recall of 96.6%. The Classification Report in Table 6.7 shows different scores for the respective classes.

Table 6.7: Classification Report SVM Subject 1

Class	Precision	Recall	f1-score	support
Blink	0.93	0.97	0.95	58
Left	0.96	0.98	0.97	45
Straight	0.99	0.99	0.99	189
Right	0.98	0.98	0.98	43
Up	0.96	0.92	0.94	49
avg/total	0.97	0.97	0.97	384

As the Confusion Matrix indicated in Figure 6.10, all classes score high in every scoring metric with an average of 97%. One class that really strengthens this classifier for Subject 1 is the

Straight class, which achieved a precision-, recall- and f1-score of 99%.

Subject 2

Figure 6.11 shows the Confusion Matrix when testing on Subject 2.

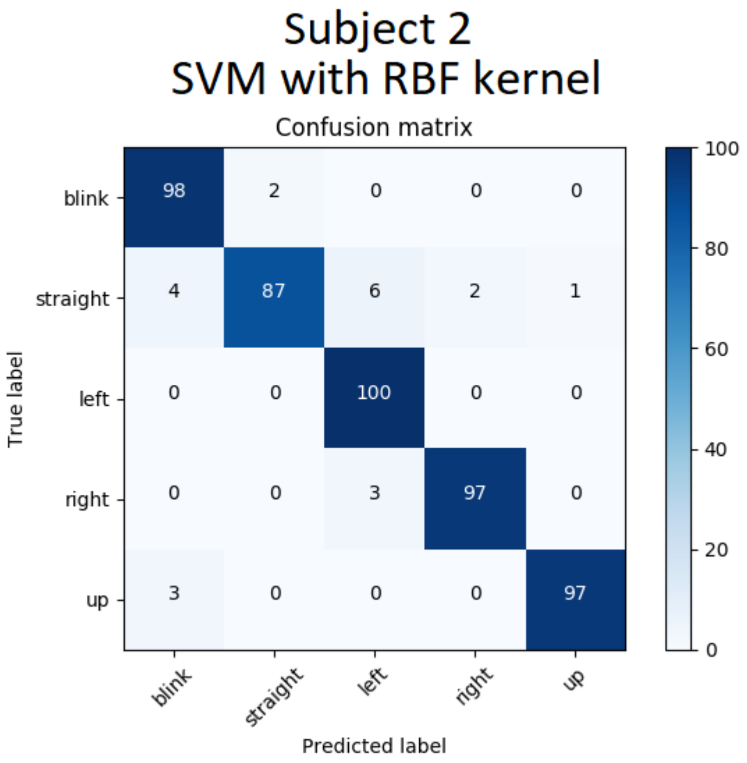


Figure 6.11: Confusion Matrix for SVM with RBF kernel while testing on Subject 2 online with extracted features. An ideal model for this test case would have 47 true positives for Blink, 122 for Straight, 28 for Left, 29 for Right and 31 for the Up class.

It can be seen that Straight is the class which confuses itself most for other classes, where all classes are represented in its confusion. The model has very high accuracy for the remaining classes with a minimum of 97% correct predictions. The overall test achieved an accuracy of 92.6 (88.7, 95.2)%, precision of 90.5% and recall of 95.6%. The Classification Report in Table 6.8 shows different scores for the respective classes.

Table 6.8: Classification Report SVM Subject 2

Class	Precision	Recall	f1-score	support
Blink	0.88	0.98	0.93	47
Left	0.78	1.00	0.88	28
Straight	0.99	0.87	0.93	122
Right	0.90	0.97	0.93	29
Up	0.97	0.97	0.97	31
avg/total	0.94	0.93	0.93	257

Table 6.8 shows that the precision for the Left class is low compared to the others. This is because Straight is more often miss-classified as Left, as shown in Figure 6.11. When comparing with Subject 1, the only weakness is that the Straight class showed a slight confusion for the other classes.

Evaluation

The SVM based model with RBF kernel was as with the offline results, the best performer for both subjects. Subject 1 was close to having no problem distinguishing the movements at all with an accuracy of 97.4 (95.3, 98.6)%, while Subject 2 showed a small weakness in the Straight class achieving an accuracy of 92.6 (88.7, 95.2)%. No other tested model was able to surpass those based on SVM with RBF kernel, for either subjects. It showed great promise for the subjects when training and testing on them respectively. Another thing to evaluate is how well the model works when trained on both subjects and tested on them respectively.

6.5.5 Unified model for SVM with RBF kernel

The unified model based on SVM with RBF kernel is trained on data from both test subjects, so there are twice as many datapoints used to train this model.

Subject 1

Figure 6.12 shows the Confusion Matrix when training on both subjects and testing on Subject 1.

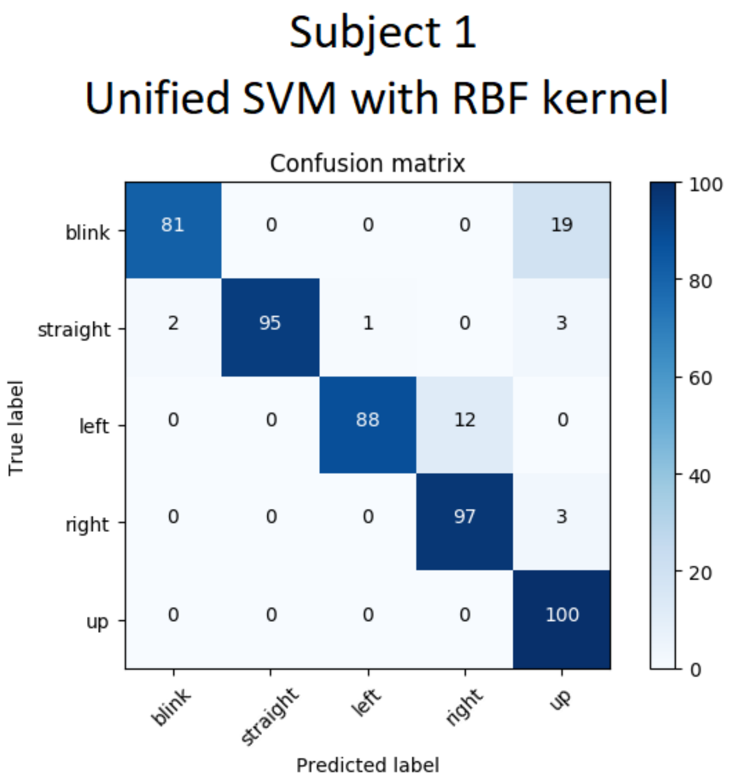


Figure 6.12: Confusion Matrix for SVM with RBF kernel while training on both subjects and testing on Subject 1 online with extracted features. An ideal model for this test case would have 32 true positives for Blink, 129 for Straight, 34 for Left, 33 for Right and 31 for the Up class.

By comparing to the specialized model for Subject 1 in Figure 6.10 it can be seen that there is a bigger confusion for the Blink and Left classes. There has also been a small decrease for the Straight and Right classes, while varying slightly for the better for the Up class. Looking at the overall performance, the model achieved an accuracy of 93.1 (89.3, 95.6)%, precision of 90.5% and recall of 92.2%. The Classification Report in Table 6.9 shows different scores for the respective classes.

Table 6.9: Classification Report SVM Subject 2

Class	Precision	Recall	f1-score	support
Blink	0.93	0.81	0.87	32
Left	0.97	0.88	0.92	34
Straight	1.00	0.95	0.97	129
Right	0.89	0.97	0.93	33
Up	0.74	1.00	0.85	31
avg/total	0.94	0.93	0.93	259

As with the Confusion Matrix in Figure 6.12 it is seen that the recall for Blink and Left are lower than the rest. The Up class also shows a lower precision because of blinks being miss predicted as Up. The overall result is, however, good with an average weighted recall of 93% and weighted precision of 94%. It would be ideal to fly the drone on the specialized model for Subject 1, but it is very possible with the unified model as well.

Subject 2

Figure 6.13 shows the Confusion Matrix when training on both subjects and testing on Subject 2.

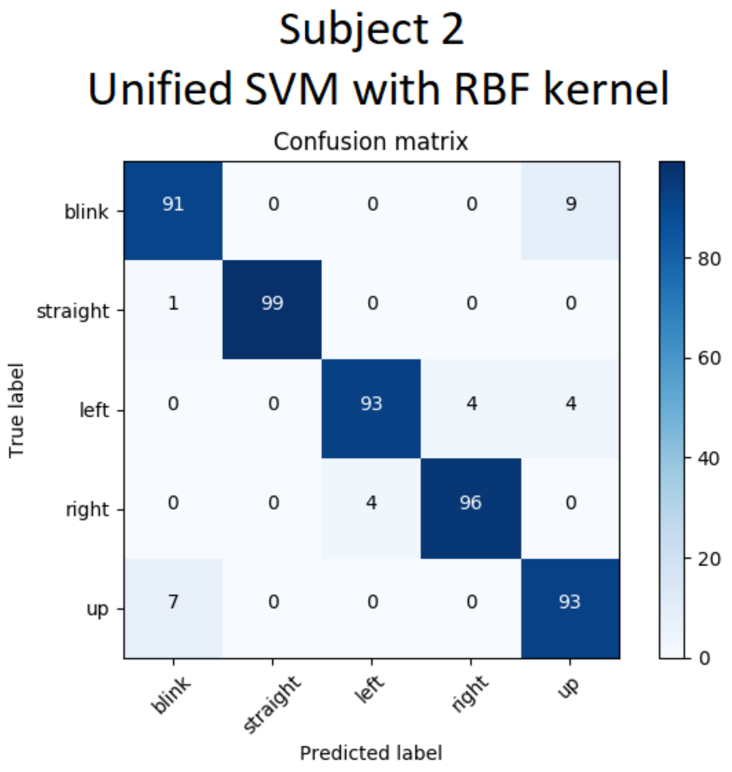


Figure 6.13: Confusion Matrix for SVM with RBF kernel while training on both subjects and testing on Subject 2 online with extracted features. An ideal model for this test case would have 45 true positives for Blink, 124 for Straight, 28 for Left, 26 for Right and 27 for the Up class.

When comparing this to both training and testing on Subject 2 in Figure 6.11 it can be seen that there are more miss-predictions for Up, Left, Right and Blink. However, Straight increased from 87% to 99% when training on both models. Subject 2 achieved an accuracy of 96.0 (92.8, 97.8)%, precision of 93.8% and recall of 94.4% when the model was trained on both subjects.

This is an increase in accuracy of 3.4% from only training on Subject 2. The Straight class is the most frequent, so an increase in correct predictions for the Straight class will have a higher impact than an increase for the other classes. The reason for the increase for the Straight class might be due to "better" Straight datapoints from Subject 1. As introduced in Section 3.3.5, the inspection of the datasets was performed by different people, meaning that the quality of the datapoints in the datasets is subjective. This can result in different classifier performance from the two datasets. The Classification Report in Table 6.10 shows different scores for the respective classes.

Table 6.10: Classification Report SVM Subject 2

Class	Precision	Recall	f1-score	support
Blink	0.93	0.91	0.92	45
Left	0.96	0.93	0.95	28
Straight	1.00	0.99	1.00	124
Right	0.96	0.96	0.96	26
Up	0.83	0.97	0.88	27
avg/total	0.96	0.96	0.96	250

As with Subject 1, Table 6.10 shows a small weakness for the Up class as both Blink and the Left class had a few miss predictions as Up (shown in Figure 6.13). The overall performance is however very good with an average weighted recall and weighted precision of 96%.

Evaluation

The SVM based models with RBF kernel, $C = 10$, $\gamma = 0.01$, $nc_m = 1.2$, no shifting of data (as introduced in Section 3.3.6) and a window length of $l_{wc} = 200$ proved to be sufficient both in the specific case when training and testing on respective subjects, and when training on both subjects. The average accuracy for both subjects from all the tests performed (both unified and separately) is 95.0% (93.6, 96.2)%. This shows that the performance is high both when creating specialized and general models. Table 6.11 shows a subject comparison when testing the SVM based models.

Table 6.11: Subject Comparison. Each cell is the accuracy achieved when trained on a subject (rows) and tested on another subject (columns).

Subject	1	2
1	97.4 (95.3, 98.6)%	N/A
2	N/A	92.6 (88.7, 95.2)%
Both	93.1 (89.3, 95.6)%	96.0 (92.8, 97.8)%

The average cross-subject accuracy achieved when training on both subjects is 94.5 (92.2, 96.2)%.

6.6 Summary and comparison with state of the art

First, the state machine with all movements considered was tested and was shown to have trouble distinguishing between classes, for both subjects. Altering the state machine to no longer consider Down ($p[i] = 2$) as a state transition from S0 to S1, but instead only use it as the "Other" and "Opposite" transitions, increased the performance for both subjects. The LinearSVC based models struggled to properly classify the blinks for both subjects but showed great promise for the other classes. While the SVM based model with RBF kernel showed good results for both subjects when training and testing respectively. Training on both subjects and testing on them respectively decreased the performance slightly for Subject 1 but increased the performance for Subject 2, having both subjects perform well on the unified model, proving that the system is capable of being used on several subjects. Table 6.12 shows a short summary of the online prediction results achieved.

Table 6.12: Summary of the online results achieved

Classifier	Training subject	Testing subject	Accuracy (%)	Features (no.)
SVM (RBF with Down)	1	1	78.8 (67.5, 86.9)	19
LinearSVC (Down)	2	2	58.9 (47.4, 69.5)	26
LinearSVC	1	1	88.3 (83.3, 92.0)	19
LinearSVC	2	2	84.4 (79.5, 88.4)	26
SVM (RBF)	1	1	97.4 (95.3, 98.6)	19
SVM (RBF)	2	2	92.6 (88.7, 95.2)	18
SVM (RBF)	Both	1	93.1 (89.3, 95.6)	23
SVM (RBF)	Both	2	96.0 (92.8, 97.8)	23

Training on both subjects and testing on them respectively, using the top performing SVM based model with RBF kernel, resulted in an average cross-subject accuracy of 94.5 (92.2, 96.2)%, average weighted recall of 94.5% and average weighted precision of 95%, which is a very good result. This is comparable to results shown in literature with similar classification schemes.

[39] achieved an accuracy of 97% for a classification scheme with horizontal movements and blinks using a K-Nearest Neighbors classifier. They also managed to achieve an average unweighted accuracy of 96.7% for all classes respectively with a SVM based classifier, where their average recall was at 99% for looking straight ahead but only between 32 to 52% (±29.6%) for blinks, right and left eye movements.

[44] achieved an average accuracy of 85.2% with a self-made classification scheme (not machine learning) for all four directional movements, straight ahead and blink.

[45] achieved an accuracy of 88.6% using a threshold-based system and a Hidden Markov Model to detect horizontal and vertical movements (straight ahead and blink excluded), the data was low-pass filtered at 7 Hz.

The closest results to this work, which was based on a SVM classifier, achieved an online accuracy of "around 95%" in [43]. This was, however, for blinks, straight, right and left movements and a bandpass filter from 0.5 to 3 Hz.

[42] used a neural network to distinguish between blinks, looking straight, to the left and right and achieved an average accuracy of 94%.

The best results found in literature was [46], which used EOG data, high pass filter at 0.05 Hz, low pass filter at 35 Hz, wavelet transform and a RBF based neural network to go beyond classifying the directions of the signal. They tried to classify eye movements in different directions, and in addition classify how large the eye movements were (in degrees). The proposed system managed to classify eye movements at ± 10° and larger, with an error rate (standard deviation) 2°. It is hard to compare the results with this paper due to the differences in the task at hand, the way of presenting the results, the different methods for processing data, extracting features and the chosen algorithm.

Table 6.13 shows a comparison of the models used and the online results in the different papers.

Table 6.13: Online accuracy comparison

	This work	[42]	[39]	[43]	[45]
Model	SVM (RBF)	Neural Network	KNN	SVM (Linear)	HMM
Classes	L,R,U,C,B	L,R,C,B	L,R,B	L,R,C,B	L,R,U,D,C
Accuracy (%)	94.5	94	97	~ 95	88.6

It is in this thesis proposed a state-machine used for controlling the drone and a model for predicting the data streaming from the scalp electrodes. It can be said to be performing well both for the specific case when training and testing on subjects respectively, and in the general case when trained on both subjects. The offline results, for the SVM based models with RBF kernel, showed lower accuracy than what was achieved in the online case. The Down class was, however, excluded from the state-machine by removing Down as a separate command. The results presented in Table 6.13 proves the effectiveness of the state machine.

Comparing with results from other literature, the proposed method for processing data, choosing a learning algorithm, finding good features, tuning the hyperparameters and the controller design can be said to be a good alternative with comparable results to methods presented in other literature.

In addition to the accuracies achieved, it was observed through the evaluation that the system is very delicate. The placement of the electrodes, disturbances in the form of muscle movements and the level of concentration all affect the signals. The "state" of the subject is therefore very important to properly control the state machine. It was observed that it in many cases was needed for the subject to "configure" into a calm state before the classifier actually made correct predictions. During this "configuration" phase, the classifier showed to frequently predict incorrectly, even when looking straight ahead. It was also observed that this differed between the subjects, where Subject 2 often needed more time to "configure" than Subject 1. As a reminder, the only filtering done in this work is at 0, 50 and 100 Hz using a notch filter, meaning that very few brain wave frequencies are filtered out. The creation of the dataset was also done in a calm state, where the subjects were focused on only making eye movements. This, together with the fact that all types of brain waves are present in the data, could be a reason for the need to be in a calm state.

Because of this delicateness and the nature of using a state-machine, in addition to training the machine learning model the subjects trained themselves to get better at using the controller to output drone commands. It is thought that better results could be achieved through more training of the subjects. However, having more data for training the model, where the data is gathered in various states of mind, could also help make for a less delicate system.

Another attempt to reduce the delicateness could be to filter out some of the higher brain wave frequencies to improve the eye movement classification, as has been done in other papers such as [43]. No obvious reason to remove these higher brain wave frequencies has been found, and information contained in these frequencies would also be discarded by doing so. In a desire to keep all the information in the signals, this filtering has not been performed. The consequence of this decision unfortunately remains unexplored in this thesis.

The mentioned literature used to compare the online results presented in this thesis concludes with the results presented for their online classification. In this work, the presented results so far is used as a proof of concept where the next step is to improve it in terms of energy efficiency. In the next chapter, execution time is used as an indicator of energy consumption for the system and the proposed features are compared and evaluated for efficiency. A trade-off between classifier performance and energy consumption is discussed to find a good solution for a battery powered device implementation. More literature showing similar results to this thesis is available and will be used as comparison in Chapter 7 to give a point of reference regarding energy efficiency.

Chapter 7

Energy Efficiency

Finding the best trade-off between performance and low energy consumption is an important aspect of implementing the proposed system on a battery powered device. This chapter presents a method for finding a feature vector that is more energy efficient than the feature vector proposed by the RFECV algorithm in Section 6.5, without too much degradation in performance. The RFECV algorithm was introduced in Section 4.1.4. It will also explore the energy consumption in terms of execution time when making predictions with different classifiers and how feature combinations affects the execution time. The results presented in this chapter is run on a private computer with a fresh install of Ubuntu 16.04 LTS where the CPU is of type: Intel Core i7-4700HQ @ 2.40GHz. Memory is 12GB DDR3 1600MHz.

7.1 Optimization strategy

There are many properties that can be used as indicators for energy consumption such as: spacial locality, regularity, number of operations and critical path [114]. Other indicators could also be the number and types of mathematical operations or doing an actual implementation on a mobile device and measure the power consumption. However, this requires a system that is a bit closer to the implementation stage than this project.

Another indicator is the execution time. Regardless of the benchmark, data load and programming language, the ratio between execution time and the energy consumption remains consistent [115]. Therefore, comparing the execution time between areas of interest in this project is an easy and illustrative way to optimize the final implemented system for lower energy consumption. When presenting the execution time, the time library [116] in the "Python Standard Library" is used. The execution time will vary depending on how much other work is performed by the OS. To work around this, a loop with 10.000 iterations is setup around the functions used to calculate specific features and make predictions and is timed for each iteration. The minimum execution times found is presented as the execution time for calculating that feature or making a prediction. Using execution times as an energy indicator has some limitations in that it is a relative comparison and can really only be compared against other solutions in the system presented in this thesis. However, many more comparisons can be done due to faster analyzation.

Lowering the execution time also fits well as a rough pre-stage to further optimize with popular energy optimization techniques, such as Dynamic Voltage Frequency Scaling (DVFS) [117] and Race-to-Halt (RTH) [118]. This is because the real-time prediction part of the system can be implemented as two independent tasks, executed at regular intervals (introduced in Section 3.1). One task fetches the raw data $\widetilde{x}[n]$, pre-processes it and store it in memory every $1/f_s = 4$ ms, and one task that does all the other processes every $1/f_p = 200$ ms. The latter might be particularly suitable to RTH optimization as it is a CPU bound task.

7.2 Energy components

There are several parts of this project that could be considered when evaluating energy efficiency. The pre-processing, classifier chosen, number of features, high and low complexity features among many other components affects the energy consumption. This work has so far focused on using machine learning to classify the different eye movements, where choosing the best features for optimal prediction has been a key part. As this has been the focus of this work so far, this will also be the area of focus when optimizing for lower energy consumption.

7.2.1 Prediction

When looking at the energy consumption for a specific model, only the part where it is used for prediction will be considered. This is because of the implementation approach, where the model could be trained on a computer where the dataset is stored, then use the pre-trained model to make predictions on the battery powered device. This way, the energy consumption when training a model is out of interest in this thesis. The classifier proposed from the online results in Section 6.6 was the SVM based model with RBF kernel and hyperparameters $C = 10$ and $\gamma = 0.01$, with a window length of $l_{w,c} = 200$.

The total complexity of making a prediction is not certain. Scikit-learn has empirically stated that "Overall you can expect the prediction time to increase at least linearly with the number of features" [119]. While [120] stated that the run-time complexity of a SVM based model with RBF kernel is given by $O(n_{sv} \times d)$, where n_{sv} is the number of support vectors and d is the feature vector dimensionality (support vectors were introduced in Section 4.4.1). On the other hand, linear SVMs is stated to have a run-time complexity of $O(d)$ [120]. This illustrates that the linear models should be unaffected by the complexity of the model (number of support vectors) when making predictions.

Scikit-learn also states that the execution time for non-linear models generally increases when the number of support vectors increases [119], which corresponds to the complexity statement made by [120]. To illustrate the difference in execution time for making a prediction with different models, all features are selected to train six models. RFECV was not used in this experiment to give a consistent comparison basis between the models. The first two are based on SVM with RBF kernel, the two next are based on the same SVM algorithm but with a linear kernel, while the last two are based on the LinearSVC algorithm proposed in Section 5.3. As a reminder, LinearSVC is built on a C-library optimized for linear classifiers [113], while the SVM based models with linear and RBF kernel are built on a C-library designed for a more general purpose [121]. Their execution times are compared in Table 7.1:

Table 7.1: Model prediction execution time comparison

Classifier	Subject	Support vectors (No.)	Execution time (μs)
SVM (RBF)	1	316	54
SVM (RBF)	2	397	63
SVM (Linear)	1	112	40
SVM (Linear)	2	261	48
LinearSVC	1	N/A	23
LinearSVC	2	N/A	26

Table 7.1 shows that linear SVM in fact is less computationally expensive than SVM with RBF kernel, and that LinearSVC outperforms the other classifiers in time used to predict. It can also be seen that the number of support vectors changes depending on the data the models are trained on, and that the execution times increases with the number of support vectors. This difference in number of support vectors can clearly be seen by comparing SVM with linear kernel for Subject 1 and 2. The reason for this might be because it is a harder classification problem to classify data from Subject 2, this is also supported by the fact that Subject 1 generally has better performance for all classes.

LinearSVC also showed a small increase in execution time from Subject 1 to Subject 2, indicating that not only the dimensionality of the feature vector matters. As shown in Table 7.1, LinearSVC outperforms SVM with RBF in terms of execution time when predicting, but as shown in Table 6.12 LinearSVC has a significantly lower performance for online prediction. The decrease in execution time is not considered large enough for the LinearSVC based model to make up for the degradation in performance. If as in this case, a non-linear classifier is needed for better performance, it is possible to reduce the execution time when predicting by minimizing the number of support vectors. Some ways of doing this could be to decrease the number of datapoints used for training, tune the hyperparameters or use features that solve the classifying problem with fewer support vectors [101]. However, as it will be shown in Section 7.2.2, the bottleneck of the system regarding execution time is the extraction of features. An attempt to reduce the number of support vectors will therefore not be done in this thesis.

7.2.2 Features

In addition to looking at the computational complexity of performing a prediction, it is also important to look at execution times when calculating the features. Some features are computationally expensive with complex operations such as FFT (spectral entropy), while others are more simplistic (min and max values). Finding the combination of features that gives the best performance metrics and the least amount of computation can save a lot of energy without losing too much accuracy. As mentioned in Section 7.2, a loop with 10.000 iterations is setup around the function calculating the specific feature and timed for each iteration. The minimum execution time is presented as the execution time for calculating that feature. Table 7.2 shows the execution time for each of the proposed features from Section 4.1.2 with a window size of $l_{w,c} = 200$:

Table 7.2: Feature execution time

Index	Name	Channel	Exec. time (μs)
0	Higuchi Fractal Dimension	1	6810
1	Higuchi Fractal Dimension	4	6861
2	Minimum Value Difference	1,3	5
3	Maximum Value Difference	1,3	5
4	Spectral Entropy	1	51
5	Spectral Entropy	4	51
6	Pearson Coefficient	3,4	64
7	Pearson Coefficient	3,4	64
8	Pearson Coefficient	1,4	64
9	Pearson Coefficient	1,4	63
10	Covariance	3,4	48
11	Covariance	3,4	48
12	Covariance	1,4	48
13	Covariance	1,4	48
14	Standard Deviation	1	22

15	Standard Deviation	4	22
16	Slope	1	7
17	Slope	4	7
18	$\theta\beta$ ratio	1	35
19	$\theta\beta$ ratio	4	35
20	Petrosian Fractal Dimension	1	101
21	Petrosian Fractal Dimension	4	101
22	Peak-to-Peak	1	5
23	Peak-to-Peak	4	5
24	Minimum Value	1	3
25	Maximum Value	1	3

Table 7.2 shows that the most expensive feature is Higuchi Fractal Dimension (HFD) with an execution time of around 6.8 ms, while the least expensive features are the minimum and maximum value of around 3 μs. If all features were chosen, the total computational cost would be the sum of the execution times as given by Equation 7.1

$$f_c = \sum_{i=0}^{n-1} f_{ci} = 14.57\text{ms}, \tag{7.1}$$

where f_{ci} is the cost of the feature with index i, n is the total number of proposed features and f_c is the total cost for calculating the feature vector.

It is important to note that around 13.6 ms of these comes from the two HFD features, so excluding these features from a feature subset would improve energy efficiency drastically.

So far, RFECV has been used to pick the best feature subset based on the accuracy of a model. As shown in Figure 7.1, the performance of the model is evaluated at every step in the iteration process. Based on these plots one can try to find a smaller number of features where the model performance has not decreased by too much.

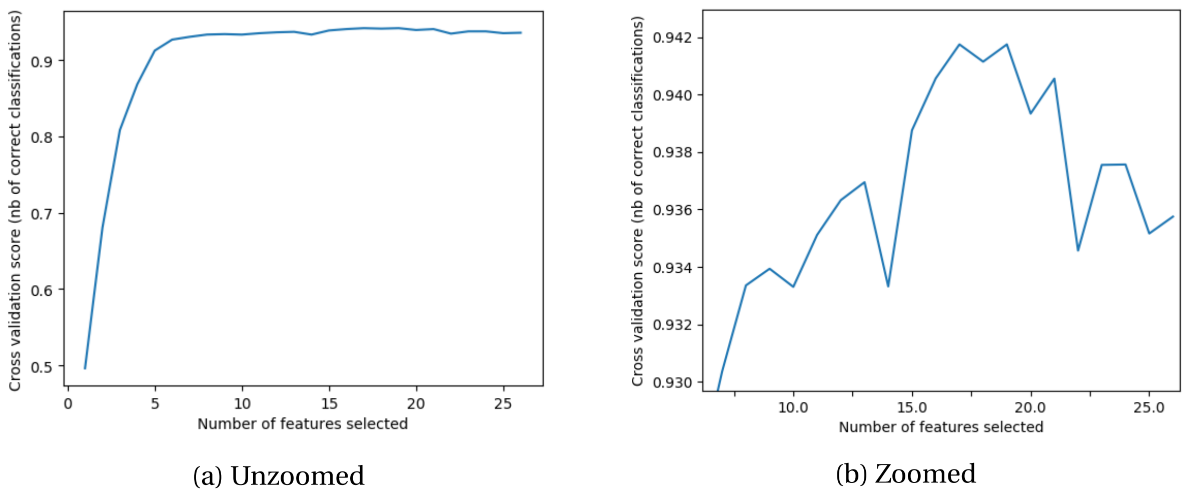


Figure 7.1: Unzoomed and zoomed RFECV plot of SVM with RBF and $l_{w,c} = 200$.

Figure 7.1 shows the RFECV plot of the SVM based model with RBF kernel, proposed in Section 6.6. The model is trained on Subject 1 with features calculated from a window length of $l_{w,c} = 200$. The optimal number of features found from RFECV is 19 with an accuracy of 94.2%. The number of support vectors is 301 for this model, resulting in a prediction execution time of 50 μs. It can be seen that it is possible to reduce the number of features to be 9 without losing too much accuracy, at 9 features the accuracy is 93.4%. There is also a noticeable increase in accuracy at 13 features, which showed to be 93.7%. As a short reminder, RFECV is a greedy algorithm which leaves out many of the possible feature subsets. As a starting point for the search of a more energy efficient feature combination, the number of features were chosen to be 9. Using 9 features over 13 has the possibility of lower power consumption and have approximately three times fewer combinations to test, making this a good choice.

With a training time of approximately 50 ms for each combination and 3.124.550 different combinations, this corresponds to around 2 days of training. This is extensive but possible to do with a brute force search and would likely be able to increase the classifier performance for a fixed number of features. The search is based on the same principle as RFECV, where the performance of the classifier is evaluated for each feature subset. There are other alternatives to a brute force search that also could work. A hill-climbing optimization algorithm such as "Simulated annealing" [122] might have given similar results with less computational time compared to the brute force search.

When performing the brute force search, each feature subset and the model performance based on several metrics in the Classification Report, as introduced in Section 4.3.1, is stored in a file so it can be accessed later. This means that the overall system can be optimized for accuracy, recall, precision, lowest possible feature execution time or some middle ground by searching through the logs for desired properties. A problem is that the brute force search is specific to one model, meaning that hyperparameters, classifier and subject used for training must be chosen in advance. The technique is therefore best used in problems where the best feature combination for a specific classifier, specific dataset and a given number of features is desired. Cross-validation (CV) is not used in this brute force search due to additional computation. The energy optimization technique using the brute force search is tested on Subject 1 to show a proof of concept. Ideally, the same technique should also be tested for Subject 2 and the unified model.

7.3 Evaluation method

When doing the brute force search for all combinations of 9 out of 26 features, all the scores for precision, recall and f1 are saved to a log file. The best performing feature subset was in this work considered to be the combination with lowest execution time with less than 1% decrease in the highest minimum value of recall. The highest minimum value of a metric is an alternative to averaging the metric, as it only takes the lowest performing class into account. Each feature combination in the logs is searched, and the model with the highest minimum recall is chosen. The distribution of minimum recall values can be seen in Figure 7.2

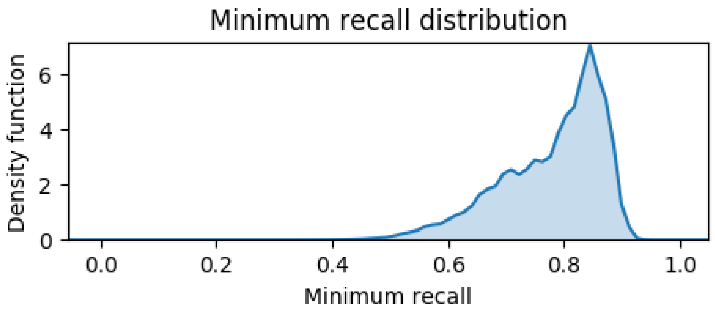


Figure 7.2: Kernel Density Estimation (KDE) [123] plot of minimum recall distribution.

We can see from Figure 7.2 that the minimum recall distribution has the highest density at 0.8 to 0.9. When analyzing the logs there were 30 possible candidates in the top 1% and of those, only 6 combinations have an execution time of less than 1 ms. The reason for choosing the highest minimum recall value is that it is most desirable to not have any poor performing classes in online prediction, as this might make the online prediction very distinct in its miss-predictions. An example where a poor performing class confuses the controller can be seen in Section 6.5.1.

As a comparison baseline, the proposed feature subset from RFECV can be seen in Table 7.3

Table 7.3: Feature execution time

Index	Name	Channel	Exec. time (μs)
0	Higuchi Fractal Dimension	1	6810
1	Higuchi Fractal Dimension	4	6861
2	Minimum Value Difference	1,3	5
3	Maximum Value Difference	1,3	5
4	Spectral Entropy	1	51
6	Pearson Coefficient	3,4	64
7	Pearson Coefficient	3,4	64
9	Pearson Coefficient	1,4	63
10	Covariance	3,4	48
11	Covariance	3,4	48
12	Covariance	1,4	48
13	Covariance	1,4	48
14	Standard Deviation	1	22
15	Standard Deviation	4	22
16	Slope	1	7
18	$\theta\beta$ ratio	1	35
22	Peak-to-Peak	1	5
24	Minimum Value	1	3
25	Maximum Value	1	3

Table 7.3 corresponds to a total execution time for extracting features of 14.21 ms. The number of support vectors is 301 for this model, with an execution time for predicting of 50 μs. Total execution time for feature extraction and prediction for this model, proposed by RFECV, is 14.26 ms. This is compared with results from the brute force search in Section 7.4.

7.4 Results

As mentioned in Section 7.3, the best performing feature subset was found by searching for the combination with lowest execution time with less than 1% decrease in the highest minimum value of recall. This feature combination, found from the brute force search, can be seen in Table 7.4.

Table 7.4: Feature execution time

Index	Name	Channel	Exec. time (μs)
4	Spectral Entropy	1	51
10	Covariance	3,4	48
12	Covariance	1,4	48
14	Standard Deviation	1	22
15	Standard Deviation	4	22
16	Slope	1	7
17	Slope	4	7
19	$\theta\beta$ ratio	4	35
24	Minimum Value	1	3

The total execution time for the feature vector, optimized for energy efficiency, in Table 7.4 is 242 μs. The number of support vectors for this model is 470, with an execution time for predicting of 5 μs. Compared to the baseline from RFECV, this is a 58x speedup in feature extraction and an increase in execution time of 4 μs for making predictions. Total execution time for feature extraction and predicting is 296 μs, giving a total speedup of 48x for both feature extraction and prediction.

Online prediction, as in Section 6.5, was performed on Subject 1 using the optimized feature vector to validate the feature combination. Figure 7.3 shows the Confusion Matrix from the test.

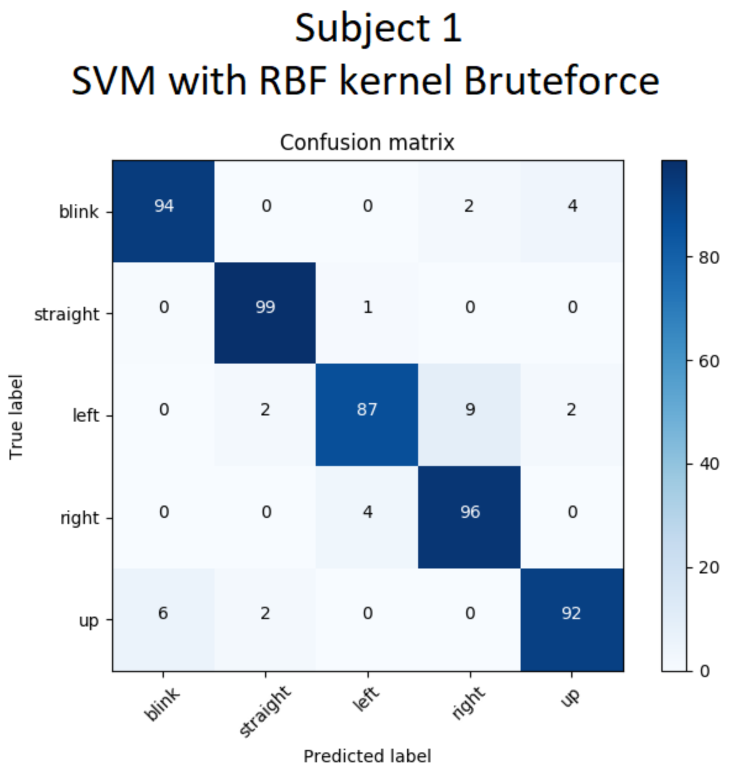


Figure 7.3: Confusion Matrix for SVM with RBF kernel while training on Subject 1 and testing on Subject 1 online with extracted features, optimized for energy efficiency. An ideal model for this test case would have 99 true positives for Blink, 211 for Straight, 53 for Left, 53 for Right and 51 for the Up class.

We can see in Figure 7.3 that there is some confusion between both Left/Right and Blink/Up. Otherwise it performs very well. The overall test achieved an accuracy of 95.3 (93.0, 96.9)%, precision of 93.0% and recall of 93.5%. The Classification Report in Table 7.5 shows different scores for the respective classes.

Table 7.5: Classification Report SVM Subject 1

Class	Precision	Recall	f1-score	support
Blink	0.96	0.94	0.95	99
Left	0.92	0.87	0.89	53
Straight	0.99	0.99	0.99	211
Right	0.88	0.96	0.92	53
Up	0.90	0.92	0.91	51
avg/total	0.95	0.95	0.95	467

7.5 Summary and comparison with state of the art

Based on the results one can see that it is possible to achieve a speedup of $48x$ for feature extraction and prediction with a 2.1% loss in accuracy for Subject 1, compared to the feature vector from RFECV. Changing the feature vector also meant an increase of 4 μs when making predictions, but this is dwarfed compared to the improvement in execution time for extracting features. Therefore, changing to the feature vector found by the brute force would result in saving considerable amounts of energy, improving the viability of implementing the system on a battery powered device.

Even though the energy optimization proved to reduce energy consumption, the proposed solution needs to be compared with similar literature to give a point of reference. Many of the articles mentioned in the online results summary in Section 6.6 did not present any form of energy evaluation or the number of features used in the feature vectors. This made it difficult to compare with the presented results in this thesis. Some articles, however, gave enough information to recreate their feature extraction process and apply them to the system in this work for comparison. Ideally, the execution time presented in this work should be compared to many of the solutions presented in other papers, but the lack of information about feature extraction methods and the number of features used made this difficult. The recreation of feature extraction processes done in other projects is time consuming, energy efficiency comparisons were therefore only done with a few papers.

In addition to the execution time when extracting the 9 optimized features, the execution time of making a prediction is compared to the different models proposed in the other papers. These will be recreated with algorithms provided by Scikit-learn, which means that comparing with papers based on neural network classifiers will not be done in this comparison. The execution times of the recreations, both for prediction and feature extraction, does not necessarily represent what actually was achieved in the systems the papers used. It does, however, show the execution time when implemented in the system made for this thesis.

The execution time comparison between the papers can be found in Table 7.6. How these execution times are obtained is explained subsequently.

[42] used 6 Auto Regression (AR) coefficients (2 for each of the 3 channels used) fed into a

neural network to distinguish between having eyes open (interpreted as looking straight ahead), blinking, looking to the left and looking to the right. Their neural network achieved a classification accuracy of 94%, which is in the same region as the performance presented in this work. To compare the energy usage, a method was found for obtaining an AR model of 2nd order in the system used in this work. The execution time was measured in the same way as with the features used in this thesis. The function for creating the second order AR model was provided by StatsModels [124] and was used on the 3 same channels as presented in the paper.

[41] was the inspiration for the use of the $\theta\beta$ ratio feature, as introduced in Section 4.1.2. They used 19 features, which consisted of the $\theta\beta$ ratio for 19 different channels (one for each). They introduced 3 different classifiers where the top performer was a Logistic Regression based model built on the liblinear C-library (same as LinearSVC), achieving an accuracy of 86%. An additional method was utilized with all 3 classifiers using a voting mechanism, increasing the accuracy to 90%, but this will not be considered in this comparison due to the additional computation introduced by having 3 classifiers. The execution time for calculating the 19 features in the system proposed in this thesis is equal to the execution time for $\theta\beta$ ratio (from Table 7.2) times 19. The results presented in the paper was during offline testing, an online implementation of the classification method was not provided.

[43] used the most related method in Section 6.6 by using a SVM based model and achieved an accuracy of "around 95%" for blinks, looking straight ahead, left and right. The method used for calculating features was Common Spatial Pattern (CSP), the number of resulting features was not mentioned. The same CSP method was tested on the system in this work with a library and dataset provided by MNE [125]. This gave an execution time of 67 μs with four classes and four channels, using a window length of $l_{w,c} = 200$. The CSP feature is quoted in Table 7.6 to have 36 features. It is debatable how one should present this because it is by definition one feature that returns a matrix with 36 elements, to be input into the classifier. This should, however, explain the discrepancy between the extraction time of 67 μs and 36 features.

[45] did not use machine learning to classify eye movements but based classification on calculating features from a method called Independent Component Analysis (ICA), which was used to decompose the signal into 5 components (frequency bands). The execution time for feature extraction is given as the time taken to extract 5 components from the original signals, which

was gathered from channel pairs (F7, F8) and (AF3, AF4). To find this execution time a function called FastICA [126], provided by Scikit-learn, was used in order to calculate the 5 components. A Hidden Markov Model was used to classify the signals, which is not a machine learning algorithm. The presented system was claimed to be a low complexity solution, which is the reason for the comparison. When implemented in the system presented in this thesis, the execution time is given from extracting 5 components from the raw signal $\widetilde{x}_{1,2,3,4}[n]$ with a window length of $l_{w,c} = 200$.

As mentioned in Section 7.2.1, the data used for training the model affects the prediction time. To do a proper comparison of prediction time would therefore mean that the datasets in the other papers should be used with the same features they proposed. This is not possible as their datasets are not available online, and an implementation of this would prove to be too demanding for the time given in this thesis. Therefore, no exact numbers regarding execution time for predictions is provided. However, to give some indication of the differences in prediction times, the different algorithms from the papers are trained and tested on the dataset provided in this thesis. This is done with the 9 features as presented in Section 7.4, showing relative execution times. None of the papers provided details regarding their models, the model implementations will therefore use the default hyperparameters as provided by Scikit-learn.

Table 7.6 shows a comparison between the different papers when implemented on the system in this thesis. The layout of Table 7.6 was inspired by [40].

Table 7.6: Execution time comparison

	This work	[42]	[41]	[43]	[45]
Model	SVM (RBF)	Neural Network	Logistic Regression	SVM (Linear)	HMM
Classes	L,R,U,C,B	L,R,C,B	L,R,U,D,C	L,R,C,B	L,R,U,D,C
Features (No.)	9	6	19	36	5
Features (type)	Mixed	Time series	Spectral	Spectral	Spectral
Feature algorithm	Mixed	AR	$\theta\beta$ ratio	CSP	ICA
Feature exec. time	242 μs	5110 μs	665 μs	67 μs	77 μs
Prediction exec. time	54 μs	N/A	22 μs	45 μs	N/A
Accuracy (%)	95 (Sub.1)	94	86	$\sim$ 95	88.6

The achieved accuracy of 95.3 (93.0, 96.9)%, presented in Table 7.6, used the features found from the brute force search in Section 7.4 and was from a specialized case where Subject 1 both trained and tested the model. The same feature vector was never tested on any other models, meaning that the accuracy presented with the lowest execution time in this work is from a specialized case, whereas the accuracy from the other papers is presented as an average over several subjects.

It was shown in this work that it was possible to get a speedup in execution time for feature extraction and prediction of 48x without losing too much performance, by testing on Subject 1. Training a model on both subjects using a feature vector from the brute force technique will likely decrease the online accuracy of 94.5%, as presented in Section 6.5, achieved with the RFECV generated feature vector. If the decrease in 2.1% accuracy for Subject 1 is considered the best possible decrease in performance, when switching from the RFECV generated feature vector to the brute force generated feature vector, the best-case accuracy for the general model would be 92.4% (more likely, less). This is not necessarily a representable number and will therefore not be presented as a performance score for the general model when using the brute force generated feature vector.

The execution time for extracting the 9 presented features from this work are somewhere in the middle, as there are solutions with execution times that are both lower and higher. The lowest execution time found in literature when using feature extraction for eye movement classification was 67 μs in [43] when testing on the system presented in this thesis. The system proposed in this thesis has an execution time of 242 μs for extracting features.

When it comes to execution time for making predictions, the model used in this thesis seems to be in the upper end of the scale, with an execution time of 54 μs for Subject 1. This could be improved with an effort on reducing the amount of support vectors but was not done in this thesis as the largest gain in execution time was found to be in the feature extraction procedure.

Regarding accuracy, the classifier presented in this thesis does very good. When comparing to classifiers with the same number of classes ([41, 45]) it has a 6.4% higher accuracy, which is a significant amount. When converting the percentages to fractions the difference becomes even clearer. The classifier in this work will, according to its accuracy, have 1 in 20 wrong predictions, while the classifier in [45] will have 1 in 8.8 wrong predictions. It is also important to mention again that this work only tested with one test subject, while [45] had 11 subjects, where several scored 100%. To classify fewer classes is a much easier classification problem, hence classifiers with fewer classes are not comparable to the one presented in this thesis [127].

Regarding both classifier performance and energy efficiency, the overall solution of this project/thesis is positive. Energy efficiency turns out to be the area with the largest potential for further improvement.

Chapter 8

Objective Evaluation, Conclusion and Recommendations for Further Work

To sum up and evaluate this thesis, the main objectives are repeated:

1. Make a system for sufficiently efficient generation of eye movement EEG data

2. Make datasets for at least two test subjects

3. Find features of the data that are fit to distinguish eye movements

4. Find a machine learning structure that, combined with these features, can distinguish eye movements

5. Implement the classification in real-time

6. Design a drone controller based on the real-time predictions from the classifier

7. Optimize the machine learning structure and selected features based on accuracy and energy efficiency

8.1 Objective evaluation

There have been many findings and results in this work. A system was created, which uses recordings from a commercially available and open source EEG headset, to classify eye movements. Methods have been presented for gathering data, finding good features of the data, finding a suitable machine learning structure, and use its output to control a drone with eye movements. A GUI was created for data gathering that instructs subjects on which eye movements to perform and when to perform them. Giving a theoretical throughput of minimum 120 datapoints per class an hour. Two datasets were created for two subjects, with a total of 2415 and 2980 datapoints, meeting Objective 1 and 2.

Recursive Feature Elimination with cross-validation was used to select features from a list of features proposed by literature search and this work. Utilizing the Scikit-learn library for Python, a Support Vector Machine classifier was trained on the datasets with selected features and showed an offline accuracy of 94% when training on both subjects, meeting Objective 3 and 4. To use the classifier in real-time, a state machine was designed, using classifier predictions to output drone commands. The state machine used a sequence of predictions to output commands, giving feedback to the user through audio stimuli. Evaluating the online performance of the classifier was done through screen-capture of predictions and commands in a terminal window using prints, together with a video-feed from a web-camera of the subject making the eye movements. The terminal and subject were recorded at the same time, allowing for a more objective evaluation of the performance. An online classification accuracy of 94.5% was achieved when training on both subjects, meeting Objective 5 and 6.

An advantage of using Recursive Feature Elimination with cross-validation was the quick implementation and proof of concept. It is however a suboptimal feature selection method due to its greedy nature. A technique was developed that used information gathered from Recursive Feature Elimination with cross-validation, to find a reduced number of features to run a brute force search on. When applying this technique, it is possible to make trade-offs between classifier accuracy and energy efficiency by comparing accuracy for different feature combinations. By using this technique, a speedup of 48x with a decrease in 2.1% online accuracy was achieved for Subject 1. This speedup can be used to reduce the energy consumption by reducing the clock frequency and voltage supply (DVFS), meeting Objective 7.

8.2 Conclusion

In this thesis a solution has been provided, with very good results regarding classification accuracy both offline and online. Selection of features and machine learning model was based on energy efficiency and accuracy, providing a good classifier with an energy efficient solution. The method used for energy optimization works well for problems similar to the ones presented in this thesis. By using this method, huge energy savings can be made with a small degradation in classifier performance. The subjects were able to use the presented system to control a drone in real time using eye movements. A video demonstration can be found here: https://www.youtube.com/watch?v=8S9dgh5TH0A.

8.3 Recommendations for further work

The system was sufficiently accurate to control a drone with a unified model. Training on data from two subjects provided positive results when testing on the subjects separately. This seems promising for creating a somewhat general model. More data from a larger number of subjects could be gathered as a larger variation of subjects could result in a good general model. If enough subjects recorded data for training, it could be possible to create a model where new test subjects, that did not participate in the making of training data, could fly the drone.

It was observed that the system is very delicate with regards to muscle movements, level of focus, other people nearby, and being in a calm state. The recording of data was done with a GUI instructing the subjects on what to do while moving nothing but the eyes, in a calm and focused state. Datapoints that showed variations due to muscular movements, bad electrode connection etc. were discarded from the training dataset as explained in Section 3.3.5. Finding a method for gathering data with varying levels of concentration and some muscle movements, could allow for a system which is not as delicate. Here, the difficulty of gathering large amounts of EEG data comes into play, but experiments regarding the delicateness of the system by introducing more variations in the data would be interesting.

Neural networks were never explored in this work but are very popular in the machine learning field. Neural nets also showed great performance when working with EOG data in [46]. Performing a proper study on neural networks has the possibility to create a neural network which could be both more energy efficient and provide better classification results. The TensorFlow Lite package [128] which was launched in the spring of 2018 could be explored, as it is a machine learning toolkit, allowing the creation of neural networks optimized for mobile and embedded devices.

The brute force search was used for energy optimization, but the proposed method was only used on Subject 1 in this work. It would be desirable to do the same optimization with models for Subject 2 and the unified model with both subjects. This would provide a better overview of classification accuracy and the energy savings that could be made, not only the savings possible for Subject 1. Doing this optimization for more test subjects might yield a general set of features, eliminating the need for optimization on future test subjects.

When using the brute force search with 9 out of 26 features, a total of 3.124.550 models were trained and tested. Performing the same search with 13 out of 26 features, a total of 10.400.600 models would be trained and tested. Ideally, each model should be evaluated with CV. With a fold of size $k = 10$, the number of models trained increases to 104.006.000, with a training time of 50 ms this corresponds to 60 days of training. This shows that the technique does not scale. In addition, the algorithm, its hyperparameters and the training data must be chosen in advance of the brute force search. Meaning that a comparison of several datasets and several hyperparameters using a brute force search should be avoided if possible. If it is desired to compare different algorithms or datasets it is suggested to find another way of selecting the feature vectors. In this work, Recursive Feature Elimination with cross-validation was used when tuning the hyperparameters and comparing the different algorithms. However, as mentioned earlier this is a suboptimal solution, and without the use of a brute force solution RFECV should be considered replaced by a hill climbing feature selection algorithm.

A more detailed study on energy optimization would be necessary before doing an actual battery powered device implementation. The pre-processing and state-machine are not analyzed for energy efficiency as a 14$^{\text{th}}$ order IIR filter and a five-layer deep if-tree is thought to be dwarfed by the feature extraction and machine learning predictions. In the long term it would be exciting to see an actual battery powered mobile device implementation and to see the system control things other than a drone.

The pre-processing used in this work filtered out frequencies around 0 Hz, 50 Hz and 100 Hz. This means that a lot of brain wave frequencies, as defined in Table 2.1, are still present in the data recorded. In this work, the focus was to classify eye movements by focusing on the peaks that also would be present in EOG recordings. It would be interesting to see a study on the possibility of using other parts than the clear peaks of the signals recorded, to classify the different eye movements.

As a feedback to the user, audio stimuli were used to call out the different commands sent to the drone. This was a simple implementation allowing for some understanding of the thought process of the drone. However, pinpointing the position of the drone through audio stimuli is difficult. It is possible when the subject is facing the drone, but otherwise not sufficient. A possible solution to this problem is to use the camera on board the drone. The possibility of implementing a point-of-view solution using virtual reality glasses to see from the drone's perspective was discussed early on. A point-of-view implementation is thought to make it much easier to intuitively control the drone. To implement this, virtual reality glasses of a smaller size is needed due to how the Ultracortex IV helmet from OpenBCI is placed on the head. If small enough virtual reality glasses are not available, it might be worth considering using a smaller EEG recording system.

The features presented in this thesis were extracted sequentially. When calculating features from this thesis, the features have the possibility to be calculated in parallel. This is achievable if implemented on an Application-Specific Integrated Circuit or FPGA. This would reduce the feature extraction execution time to that of the feature with the longest execution time.

Bibliography

[1] Hans Berger. Uber das elektrenkephalogramm des menschen. *Arch. Psychiatr Nervenkr*, 1, 1929.

[2] Sarah N Abdulkader, Ayman Atia, and Mostafa-Sami M Mostafa. Brain computer interfacing: Applications and challenges. *Egyptian Informatics Journal*, 16(2):213–230, 2015.

[3] Sumeet Agarwal. Machine learning: A very quick introduction, 2013. URL `http://web.iitd.ac.in/~sumeet/mlintro_doc.pdf`. Visited 23.05.18.

[4] Christopher M Bishop. Machine learning and pattern recognition. *Information Science and Statistics. Springer, Heidelberg*, 2006.

[5] Adrian Ribe and Kristian Husevåg Krohn. Real-time pre-processing of eeg data for machine learning, using commercial grade equipment to control a drone. *TFE4520 Design of Digital Systems - Specialization project, NTNU*, 2017.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Ramesh Srinivasan Paul L. Nunez. *Electric Fields of The Brain, The Neurophysics of EEG.* Oxford, 2006.

[8] Jasmine Poteat. The nervous system, 2011. URL `https://sites.google.com/site/anatomybody/calendar`. Visited 18.10.2017.

[9] A. R. Luria. *Higher Cortical Functions in Man.* New York: Basic Books, 1966.

[10] Joaquin M Fuster. Frontal lobe and cognitive development. *Journal of Neuro cytology*, 1, 2002.

[11] Henry Gray. *Anatomy of the Human Body*. Lea and Febiger, 1918.

[12] Korbininan Broadmann. *Vergleichende Lokalisationslehre der Großhirnrinde : in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Verlag von Johann Ambrosius Barth, 1909.

[13] Dr. K. Brodmann and Laurence J. Garey. *BRODMANN'S Localisation in the Cerebral Cortex*, volume 1. Springer, 2006.

[14] Anja K.E.Horn and R. John Leigh. *Handbook of Clinical Neurology*, volume 102. ELSEVIER, 2011.

[15] Kamel Nidal and Aamir Saeed Malik. *EEG/ERP Analysis: Methods and Applications*, volume 1. CRC Press, 2014.

[16] H. Aurlien*, I.O. Gjerde, J.H. Aarseth, G. Eldøen, B. Karlsen, H. Skeidsvoll, and N.E. Gilhus. Eeg background activity described by a large computerized database. *Elsevier*, 1, 2003.

[17] Patrick Berg and Michael Scherg. Dipole models of eye movements and blinks. *Electroencephalography and clinical Neurophysiology*, 79(1):36–44, 1991.

[18] Lars-Erik Notevarp Bjørge and Trond Hübertz Emaus. Identification of eeg-based signature produced by visual exposure to the primary colors rgb. *Master's thesis NTNU*, 2017.

[19] Sarah N.Abdulkader, Ayman Atia, and Mostafa-Sami M.Mostafa. *Brain computer interfacing: Applications and challenges*, volume 1. ELSEVIER, 2015.

[20] Pedro Ponce, Arturo Molina, David C Balderas, and Dimitra Grammatikou. Brain computer interfaces for cerebral palsy. In *Cerebral Palsy-Challenges for the Future*. InTech, 2014.

[21] John Karat and Jean Vanderdonckt. Human-computer interaction series. *Editor Springer London*, 2010.

[22] Tristan Dane Cloyd. *(r) Evolution in Brain-Computer Interface Technologies for Play:(non) Users in Mind.* PhD thesis, Virginia Polytechnic Institute and State University, 2013.

[23] Rajesh PN Rao and Reinhold Scherer. Brain-computer interfacing [in the spotlight]. *IEEE Signal Processing Magazine*, 27(4):152–150, 2010.

[24] Microsoft. Microsoft kinect, 2018. URL https://msdn.microsoft.com/en-us/library/hh438998.aspx. Visited 13.04.2018.

[25] Microsoft. Microsoft hololens, 2018. URL https://www.microsoft.com/nb-no/hololens. Visited 13.04.2018.

[26] NeuroSky. Neurosky, 2018. URL http://neurosky.com/. Visited 13.04.2018.

[27] Emotiv. Emotiv, 2018. URL https://www.emotiv.com/. Visited 13.04.2018.

[28] OpenBCI. Openbci, 2018. URL http://openbci.com/. Visited 13.04.2018.

[29] Scikit-learn. An introduction to machine learning with scikit-learn, 2017. URL http://scikit-learn.org/stable/tutorial/basic/tutorial.html. Visited 14.05.2018.

[30] Ethen Alpaydin. *Machine Learning: The New AI*, volume 1. MIT Press, 2016.

[31] Iliya Valchanov. Machine learning: An overview, 2018. URL https://www.datascience.com/blog/machine-learning-overview. Visited 14.05.2018.

[32] Jonathan Shewchuk. Cs 189/289a introduction to machine learning, 2017. URL https://people.eecs.berkeley.edu/~jrs/189/lec/02.pdf. Visited 03.04.2018.

[33] Isabelle Guyon and André Elisseeff. An introduction to feature extraction. In *Feature extraction*, pages 1–25. Springer, 2006.

[34] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml. Visited 13.03.2018.

[35] The Scikit learn developers. Plot different svm classifiers in the iris dataset, 2017. URL http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html. Visited 25.05.2018.

[36] Gary Ericson and William Anton Rohm. How to choose algorithms for microsoft azure machine learning, 2018. URL https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice. Visited 26.02.2018.

[37] Anil K Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.

[38] Sarunas J Raudys, Anil K Jain, et al. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on pattern analysis and machine intelligence*, 13(3):252–264, 1991.

[39] Ker-Jiun Wang, Lan Zhang, Bo Luan, Hsiao-Wei Tung, Quanfeng Liu, Jiacheng Wei, Mingui Sun, and Zhi-Hong Mao. Brain-computer interface combining eye saccade two-electrode eeg signals and voice cues to improve the maneuverability of wheelchair. In *Rehabilitation Robotics (ICORR), 2017 International Conference on*, pages 1073–1078. IEEE, 2017.

[40] Chi-Hsuan Hsieh and Yuan-Hao Huang. Low-complexity eeg-based eye movement classification using extended moving difference filter and pulse width demodulation. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 7238–7241. IEEE, 2015.

[41] Sherif M Abdelfattah, Kathryn E Merrick, and Hussein A Abbass. Eye movements as information markers in eeg data. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–7. IEEE, 2016.

[42] Hai Thanh Nguyen, Nguyen Trung, Vo Toi, and Van-Su Tran. An autoregressive neural network for recognition of eye commands in an eeg-controlled wheelchair. In *Advanced Technologies for Communications (ATC), 2013 International Conference on*, pages 333–338. IEEE, 2013.

[43] Soumya Sen Gupta, Sumit Soman, P Govind Raj, Rishi Prakash, S Sailaja, and Rupam Borgohain. Detecting eye movements in eeg for controlling devices. In *Computational Intelligence and Cybernetics (CyberneticsCom), 2012 IEEE International Conference on*, pages 69–73. IEEE, 2012.

[44] Abdelkader Nasreddine Belkacem, Duk Shin, Hiroyuki Kambara, Natsue Yoshimura, and Yasuharu Koike. Online classification algorithm for eye-movement-based communication systems using two temporal eeg sensors. *Biomedical Signal Processing and Control*, 16:40–47, 2015.

[45] Chi-Hsuan Hsieh, Hao-Ping Chu, and Yuan-Hao Huang. An hmm-based eye movement detection system using eeg brain-computer interface. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 662–665. IEEE, 2014.

[46] Rafael Barea, Luciano Boquete, Sergio Ortega, Elena López, and JM Rodríguez-Ascariz. Eog-based eye movements codification for human computer interaction. *Expert Systems with Applications*, 39(3):2677–2683, 2012.

[47] J. Philipp de Graaff. Ps-drone, 2014. URL http://www.playsheep.de/drone/. Visited 17.04.2018.

[48] OpenBCI. Ultracortex mark iv, 2015. URL http://docs.openbci.com/Headware/01-Ultracortex-Mark-IV. Visited 16.10.2017.

[49] OpenBCI. Cyton, 2015. URL http://docs.openbci.com/Hardware/02-Cyton. Visited 16.10.2017.

[50] Texas Instruments. Ads1299 datasheet, 2017. URL http://www.ti.com/lit/ds/symlink/ads1299.pdf. Visited 16.10.2017.

[51] OpenBCI. Cytonpictures, 2017. URL https://shop.openbci.com/collections/frontpage/products/cyton-biosensing-board-8-channel?variant=38958638542. Visited 26.10.2017.

[52] OpenBCI. Openbci github, 2017. URL https://github.com/OpenBCI/OpenBCI_Python. Visited 18.10.2017.

[53] Florida Research. Disposable / reusable dry eeg electrode [tde-200], 2017. URL https://fri-fl-shop.com/product/tde-200/. Visited 16.10.2017.

[54] Ralph Beebe Blackman and John W Tukey. The measurement of power spectra, 1958. Dover Publications Inc.

[55] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*, volume 1. IEEE Computer Society, 1997.

[56] George Arfken. Discrete orthogonality–discrete fourier transform, 1985. Academic Press.

[57] Marius Hart. 10/20 figure, 2008. URL http://www.mariusthart.net/downloads/eeg_electrodes_10-20.svg. Visited 23.10.2017.

[58] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[59] SciPy. scipy.signal.savgol_filter, 2015. URL https://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.signal.savgol_filter.html. Visited 12.03.2018.

[60] Inc Statsoft. Electronic statistics textbook, 2013. URL http://www.statsoft.com/textbook/. Visited 04.03.2018.

[61] Pornchai Phukpattaranont, Siriwadee Aungsakul, Angkoon Phinyomark, and Chusak Limsakul. Efficient feature for classification of eye movements using electrooculography signals. *Thermal Science*, 20(suppl. 2):563–572, 2016.

[62] Shayan Motamedi-Fakhr, Mohamed Moshrefi-Torbati, Martyn Hill, Catherine M Hill, and Paul R White. Signal processing techniques applied to human sleep eeg signals—a review. *Biomedical Signal Processing and Control*, 10:21–33, 2014.

[63] M Hernán Díaz, Felisa M Córdova, Lucio Cañete, Fredi Palominos, Fernando Cifuentes, Crystian Sánchez, and Matías Herrera. Order and chaos in the brain: fractal time series analysis of the eeg activity during a cognitive problem solving task. *Procedia Computer Science*, 55:1410–1419, 2015.

[64] Tomoyuki Higuchi. Approach to an irregular time series on the basis of the fractal theory. *Physica D: Nonlinear Phenomena*, 31(2):277–283, 1988.

[65] W Klonowski, E Olejarczyk, and R Stepien. Sleep-eeg analysis using higuchi's fractal dimension. In *Proceedings of the International Symposium on Nonlinear Theory and Its Applications (NOLTA'05)*, pages 222–225, 2005.

[66] MTR Peiris, RD Jones, PR Davidson, PJ Bones, and DJ Myall. Fractal dimension of the eeg for detection of behavioural microsleeps. In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 5742–5745. IEEE, 2006.

[67] Forrest Sheng Bao, Xin Liu, and Christina Zhang. Pyeeg: an open source python module for eeg/meg feature extraction. *Computational intelligence and neuroscience*, 2011, 2011.

[68] Douglas G Altman and J Martin Bland. Standard deviations and standard errors. *Bmj*, 331 (7521):903, 2005.

[69] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[70] Jun D Bonitaa, Alfonso M Albanob, Leo Cristobal C Ambolode IIa, and Paul E Rappc. Study of correlations of multichannel human eeg. In *Proceedings of the 11th SPVM National Physics Conference and Workshop. sl: sn*, pages 31–33, 2009.

[71] Bin Yang and Aihua Zhang. Power spectral entropy analysis of eeg signal based-on bci. In *Control Conference (CCC), 2013 32nd Chinese*, pages 4513–4516. IEEE, 2013.

[72] PR Bartel, FJ Smith, and PJ Becker. A comparison of eeg spectral entropy with conventional quantitative eeg at varying depths of sevoflurane anaesthesia. *Southern African Journal of Anaesthesia and Analgesia*, 11(3):89–93, 2005.

[73] R Quian Quiroga, S Blanco, OA Rosso, H Garcia, and A Rabinowicz. Searching for hidden information with gabor transform in generalized tonic-clonic seizures. *Electroencephalography and clinical Neurophysiology*, 103(4):434–439, 1997.

[74] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[75] Olivier Chapelle and S Sathiya Keerthi. Multi-class feature selection with support vector machines. In *Proceedings of the American statistical association*, 2008.

[76] The Scikit learn developers. Recursive feature elimination with cross-validation, 2018. URL http://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_with_cross_validation.html#sphx-glr-auto-examples-feature-selection-plot-rfe-with-cross-validation-py. Visited 19.04.2018.

[77] The Scikit learn developers. Preprocessing data, 2017. URL http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler. Visited 15.05.2018.

[78] The Scikit learn developers. sklearn.preprocessing.standardscaler, 2017. URL http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Visited 02.06.2018.

[79] Katrina Wakefield. A guide to machine learning algorithms and their applications, N/A. URL https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html#. Visited 26.02.2018.

[80] Vladimir Cherkassky and Yunqian Ma. Practical selection of svm parameters and noise estimation for svm regression. *Neural networks*, 17(1):113–126, 2004.

[81] The Scikit learn developers. Choosing the right estimator, 2017. URL http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html. Visited 15.05.2018.

[82] The Scikit learn developers. Confusion matrix, 2018. URL http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py. Visited 19.04.2018.

[83] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[84] Vincent Van Asch. Macro-and micro-averaged evaluation measures [[basic draft]]. *Belgium: CLiPS*, 2013.

[85] The Scikit learn developers. Preprocessing data, 2017. URL http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html. Visited 24.05.2018.

[86] The Scikit learn developers. Cross-validation: evaluating estimator performance, 2017. URL http://scikit-learn.org/stable/modules/cross_validation.html. Visited 15.05.2018.

[87] Christian Gold. *Modeling of Take-Over Performance in Highly Automated Vehicle Guidance*. PhD thesis, Universität München, 2016.

[88] Rokach Lior et al. *Data mining with decision trees: theory and applications*, volume 81. World scientific, 2014.

[89] Wil MP Van der Aalst, Vladimir Rubin, HMW Verbeek, Boudewijn F van Dongen, Ekkart Kindler, and Christian W Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1):87, 2010.

[90] Qingkai Kong. Machine learning 9 - more on artificial neural network, 2017. URL http://qingkaikong.blogspot.no/2017/02/machine-learning-9-more-on-artificial.html. Visited 11.02.2018.

[91] Fabien Lotte, Marco Congedo, Anatole Lécuyer, Fabrice Lamarche, and Bruno Arnaldi. A review of classification algorithms for eeg-based brain–computer interfaces. *Journal of neural engineering*, 4(2):R1, 2007.

[92] Manoj Thulasidas, Cuntai Guan, and Jiankang Wu. Robust classification of eeg signal for brain-computer interface. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 14(1):24–29, 2006.

[93] Joar Molvær and Fredrik Worren. A unified real-time feature extraction and classification process for a bci based on empirical mode decomposition and support vector machine. *Master's thesis NTNU*, 1, 2016.

[94] Cuntai Guan, Xiaoyuan Zhu, S Ranganatha, M Thulasidas, and Jiankang Wu. Robust classification of event-related potential for brain-computer interface. In *Int. Conf. Adv. Medical Signal Info. Processing*, pages 321–326, 2004.

[95] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[96] Morgun Ivan. How-to simulate support vector machine (svm) in r, 2014. URL http://en.proft.me/2014/04/22/how-simulate-support-vector-machine-svm-r/. Visited 12.03.2018.

[97] Rashmi Jain. Simple tutorial on svm and parameter tuning in python and r, 2017. URL https://www.hackerearth.com/blog/machine-learning/simple-tutorial-svm-parameter-tuning-python-r/. Visited 20.03.2018.

[98] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[99] S Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

[100] The Scikit learn developers. Support vector machine kernel functions, 2017. URL http://scikit-learn.org/stable/modules/svm.html#kernel-functions. Visited 15.05.2018.

[101] The Scikit learn developers. Support vector machines, 2018. URL http://scikit-learn.org/stable/modules/svm. Visited 23.04.2018.

[102] Trevor Hastie, Saharon Rosset, Robert Tibshirani, and Ji Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5(Oct):1391–1415, 2004.

[103] The Scikit learn developers. Decision trees, 2017. URL http://scikit-learn.org/stable/modules/tree.html. Visited 27.05.2018.

[104] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[105] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009. doi: 10.4249/scholarpedia.1883. revision #136646.

[106] Parrot. Ar.drone 2.0, 2018. URL `https://www.parrot.com/global/drones/parrot-ardrone-20-elite-edition#parrot-ardrone-20-elite-edition`. Visited 17.04.2018.

[107] Peter Christ Michael Adams Christian Menßen Timm Hörmann, Marc Hesse and Ulrich Rückert. Fine-grained prediction of cognitive workload in a modern working environment by utilizing short-term physiological parameters. In *Biomedical Engineering Systems and Technologies 9th International Joint Conference, BIOSTEC 2016, Rome, Italy, February 21–23, 2016, Revised Selected Papers*, page 214, 2016.

[108] Mark Hammond. pywin32, 2018. URL `https://github.com/mhammond/pywin32`. Visited 06.05.2018.

[109] eSpeak. espeak text to speech, 2015. URL `http://espeak.sourceforge.net/`. Visited 06.05.2018.

[110] IEEE. Ieee std 1003.1-2017, 2017. URL `https://standards.ieee.org/findstds/standard/1003.1-2017.html`. Visited 06.05.2018.

[111] OBS Studio Contributors. Open broadcaster software, 2018. URL `https://obsproject.com`. Visited 28.04.2018.

[112] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[113] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[114] Jan Rabaey, Lisa Guerra, and Renu Mehra. Design guidance in the power dimension. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 5, pages 2837–2840. IEEE, 1995.

[115] Luis Corral, Anton B Georgiev, Alberto Sillitti, and Giancarlo Succi. Can execution time describe accurately the energy consumption of mobile apps? an experiment in android. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, pages 31–37. ACM, 2014.

[116] Python Software Foundation. Time library for python 2.7, 2018. URL `https://docs.python.org/2/library/time.html`. Visited 20.05.2018.

[117] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 38–43, 2007.

[118] David C Snowdon12, Etienne Le Sueur, Stefan M Petters, and Gernot Heiser. A platform for os-level power management. *The European Professional Society on Computer Systems 2009*, 2009.

[119] The Scikit learn developers. Computational performance, 2018. URL `http://scikit-learn.org/stable/modules/computational_performance.html`. Visited 23.04.2018.

[120] Marc Claesen, Frank De Smet, Johan AK Suykens, and Bart De Moor. Fast prediction with svm models containing rbf kernels. *arXiv preprint arXiv:1403.0736*, 2014.

[121] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[122] S. Kirkpatrick; C. D. Gelatt; M. P. Vecchi. Optimization by simulated annealing. In *Science, New Series, Vol. 220, No. 4598.*, pages 671–680, 1983.

[123] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *Ann. Math. Statist.*, 27(3):832–837, 09 1956.

[124] Jonathan Taylor statsmodels-developers Josef Perktold, Skipper Seabold. Statsmodels, statistics in python, 2013. URL http://www.statsmodels.org/0.6.1/generated/statsmodels.tsa.ar_model.AR.fit.html#statsmodels.tsa.ar_model.AR.fit. Visited 10.05.2018.

[125] Alexandre Gramfort, Martin Luessi, Eric Larson, Denis Engemann, Daniel Strohmeier, Christian Brodbeck, Roman Goj, Mainak Jas, Teon Brooks, Lauri Parkkonen, and Matti Hämäläinen. Meg and eeg data analysis with mne-python. *Frontiers in Neuroscience*, 7:267, 2013. ISSN 1662-453X. doi: 10.3389/fnins.2013.00267. URL https://www.frontiersin.org/article/10.3389/fnins.2013.00267.

[126] The Scikit learn developers. sklearn.decomposition.fastica, 2018. URL http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html#sklearn.decomposition.FastICA. Visited 23.05.2018.

[127] Daniel Silva-Palacios, Cèsar Ferri, and María José Ramírez-Quintana. Improving performance of multiclass classification by inducing class hierarchies. *Procedia Computer Science*, 108:1692–1701, 2017.

[128] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Vistited 15.05.18.

[129] Julius O. Smith. *Introduction to Digital Filters: with Audio Applications*. W3K Publishing, 2007.

[130] Scipy. scipy.signal.iirnotch, 2017. URL https://docs.scipy.org/doc/scipy-0.19.1/reference/generated/scipy.signal.iirnotch.html. Visited 09.11.2017.

[131] Per Ahlgren, Bo Jarneving, and Ronald Rousseau. Requirements for a cocitation similarity measure, with special reference to pearson's correlation coefficient. *Journal of the Association for Information Science and Technology*, 54(6):550–560, 2003.

Appendix A

Pre-processing

Pre-processing is needed in order to remove undesired parts of the signal, and make sure that the data fed into the classifier contains the desired information. The method for processing data was proposed in a previous project [5] and is not considered an essential part of this thesis but serves as additional information to the curious reader or if one wishes to recreate parts of this work in the future. This Appendix will shortly summarize the results from that project.

As a starting point we can look at some time-series and spectrum plots in Figure A.1.

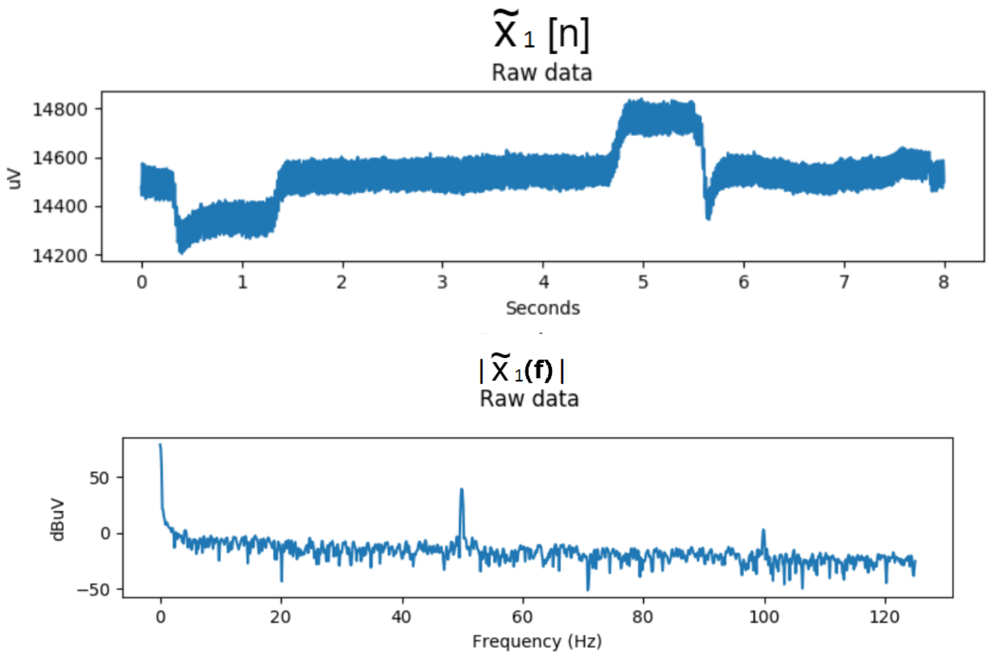


Figure A.1: Time-series of $\widetilde{x}_1[n]$ and amplitude response $|\widetilde{x}_1(f)|$.

From Figure A.1 we can deduct some desired filter properties:

- Removal of DC offset and other disturbances
- Delay as short as possible, not over 500 ms
- Step response with short fall time, low amount of disturbance and short settling time

Both the DC component and 50 Hz noise are considered to be technical artifacts and are therefore regarded as parasitic [15]. Delay should ideally be as short as possible as the drone is going to be controlled in real time, which means that a large delay makes for unresponsive control. Having a step response with short fall time and short settling time means that there can be many events in quick succession, without having them overlap. It also means that it is clearer what part of the waveform that is a direct response to the step, caused by the filter. Each filter tested in the previous project [5] was given a sum of points based on the desired properties.

The results showed that the filter with the highest score implemented notch filters at 0 Hz, 50 Hz and 100 Hz and was named the "septa notch filter".

Figure A.2 shows the amplitude response $|\widetilde{x}_{sn,1}(f)|$ of the output signal from the septa notch filter.

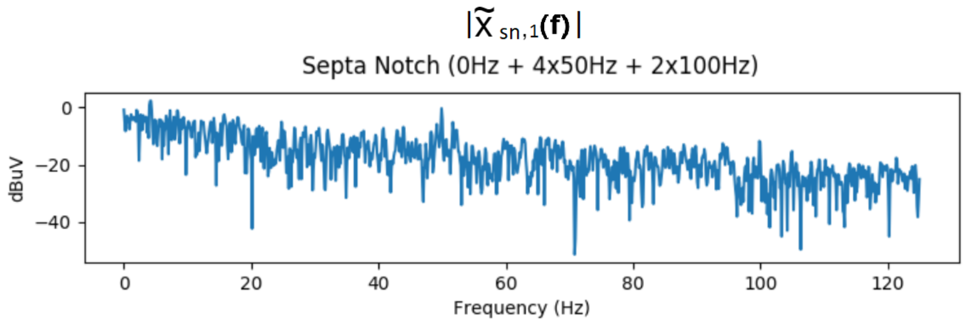


Figure A.2: Amplitude response $|\widetilde{x}_{sn,1}(f)|$.

Figure A.3 show the time-series plot of $\widetilde{x}_{sn,1}[n]$.

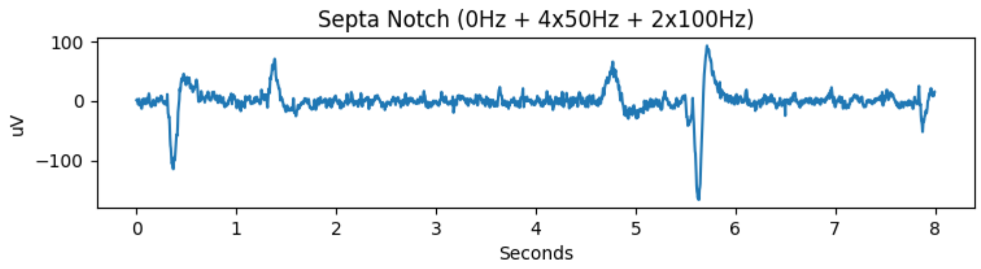


Figure A.3: Time-series plot of $\widetilde{x}_{sn,1}(f)$

The septa notch filter uses a first order notch filter to remove a DC component, an eight-order notch filter at 50 Hz and a fourth order notch filter at 100 Hz. A block diagram can be seen in figure A.4.

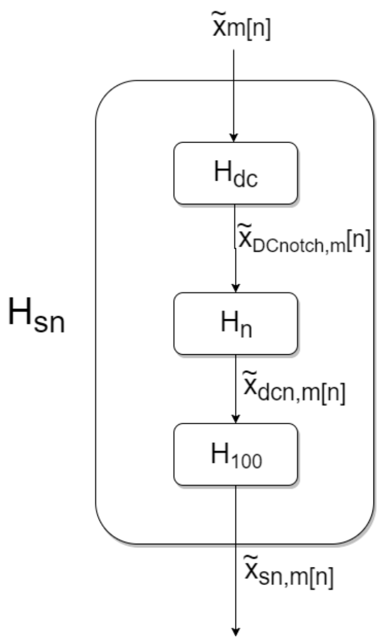


Figure A.4: Block diagram of the septa notch filter $H_{sn}(f)$ with a 0 Hz notch filter, 50 Hz notch filter and a 100 Hz notch filter.

$H_{dc}(f)$ is a first order notch filter at 0 Hz, $H_n(f)$ is an eight order notch filter at 50 Hz, and $H_{100}(f)$ is a fourth order notch filter at 100 Hz. The first order notch filter $H_{dc}(f)$ coefficients are obtained from the DC blocker chapter in the "Introduction to Digital Filters: with Audio Applications" [129] book and shown in table A.1.

Table A.1: DC notch filter coefficients[129]

index	a_j	b_i
0	1	1
1	-0.9	-1

The eight order notch filter $H_n(f)$ at 50 Hz is made by cascading four of the second order 50 Hz notch filters presented in table A.2.

Table A.2: 50 Hz notch filter coefficients

index	a_j	b_i
0	1	0.9876
1	0.6104	-0.6104
2	0.9752	-0.9876

The 100 Hz notch filters is made by cascading two second order IIR filters designed with a Q factor of 50, which is implemented with the SciPy function iirnotch [130]. The filter coefficients can be seen in Table A.3.

Table A.3: 100 Hz notch filter coefficients

index	a_j	b_i
0	1	0.97
1	1.57	1.57
2	0.95	0.97

When we combine the filter coefficients the results can be found in Table A.4.

Table A.4: Combined filter coefficients

index	a_j	b_i
0	1	0.9051
1	-0.1847	-0.2136
2	2.1783	1.9424
3	1.0537	0.8235
4	2.0830	1.7287
5	0.4115	-0.0816
6	2.5758	2.1561
7	-1.5179	-2.1561
8	0.5387	0.0816
9	-1.2193	-1.7287
10	-0.5383	-0.8235
11	-1.5404	-1.9424
12	0.2172	0.2136
13	-0.7360	-0.9051

The filtering is performed with the general IIR filter equation found in Equation A.1

$$x_m[n] = \frac{1}{a_0}\left(\sum_{i=0}^{P} b_i \tilde{x}_m[n-i] - \sum_{j=1}^{R} a_j x_m[n-j]\right) \tag{A.1}$$

Where:

- $P = 14$ is the feedforward filter order
- $R = 13$ is the feedback filter order

The amplitude- impulse-, step-, and phase-response of the resulting septa notch filter is shown in Figure A.5.

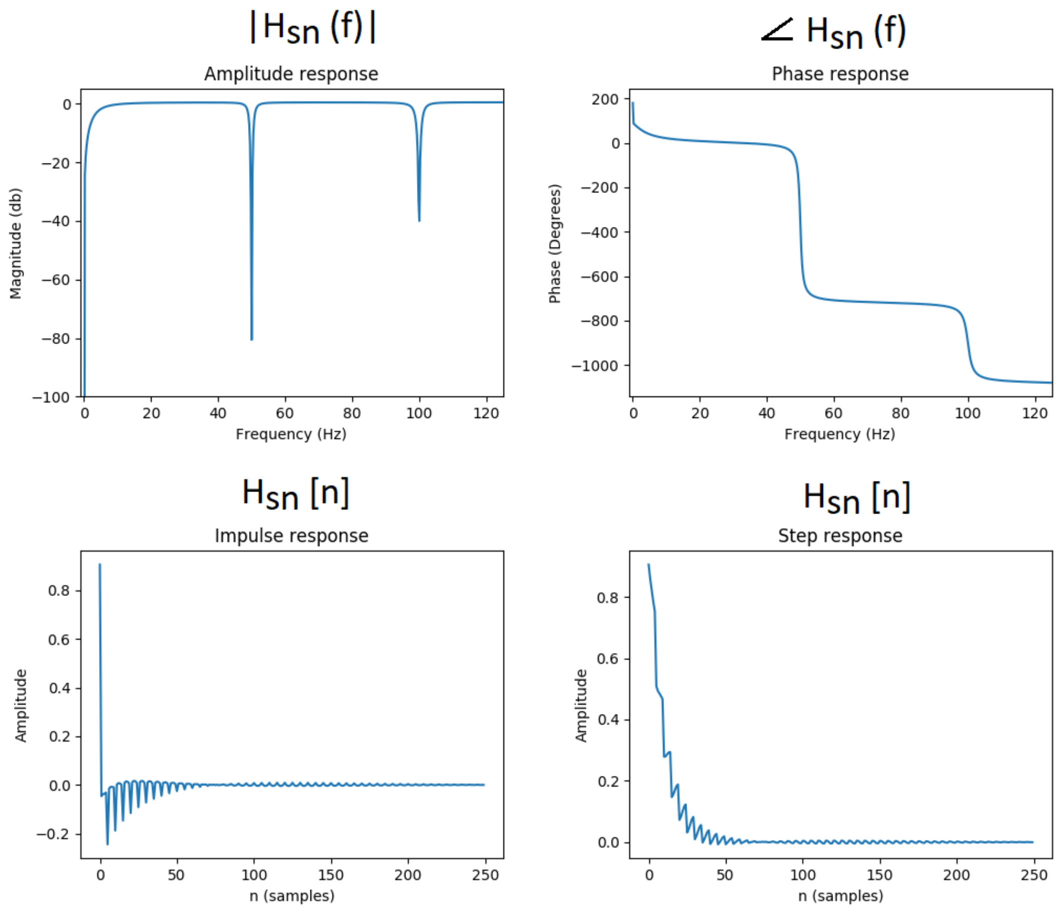


Figure A.5: Showing the amplitude-, impulse-, phase- and step-response of the septa notch filter.

Appendix B

Dataset class examples

The following Appendix shows plots of various eye movements that are contained in the dataset. Plots like these are what was used when inspecting the quality of the different data points.

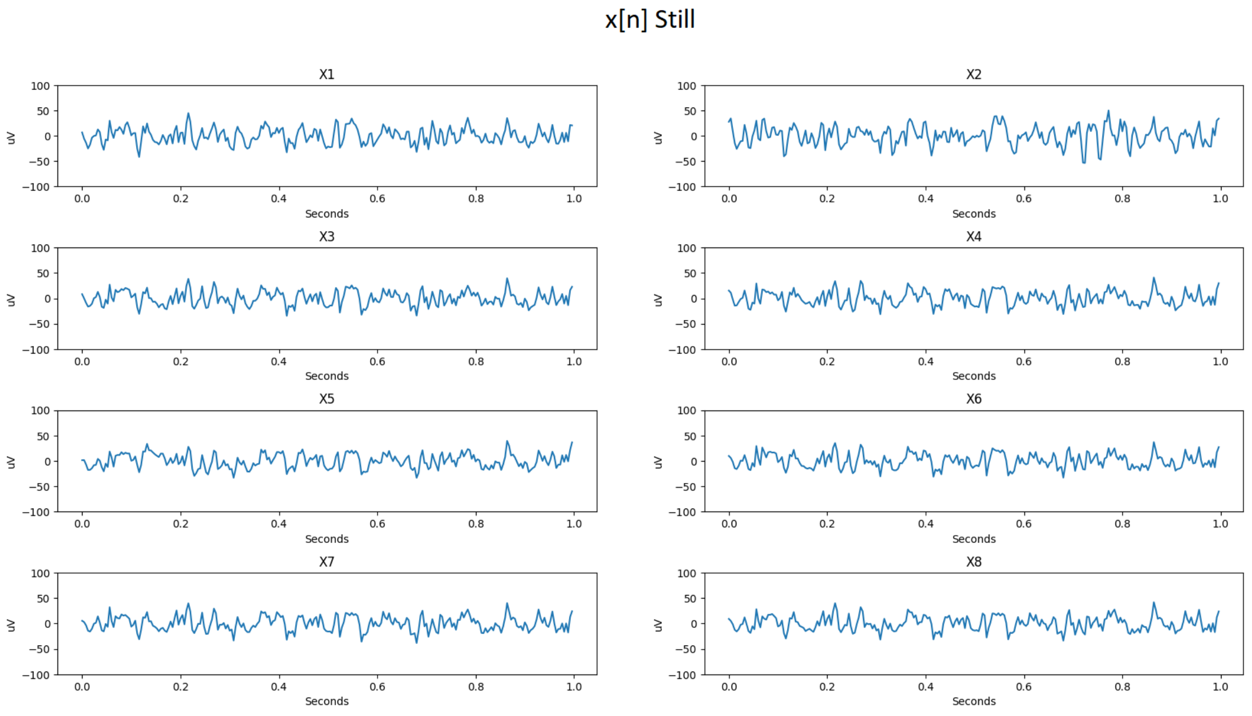


Figure B.1: Staring straight forward.

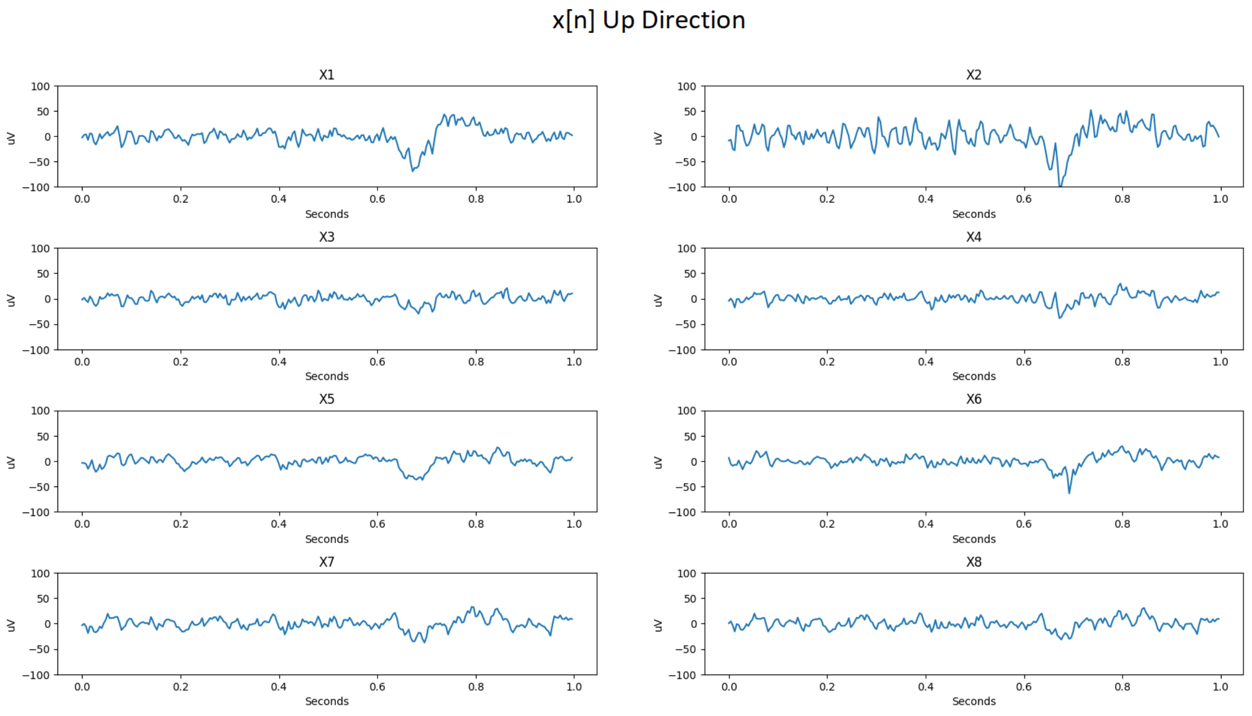


Figure B.2: Up from center at 0.7 seconds.

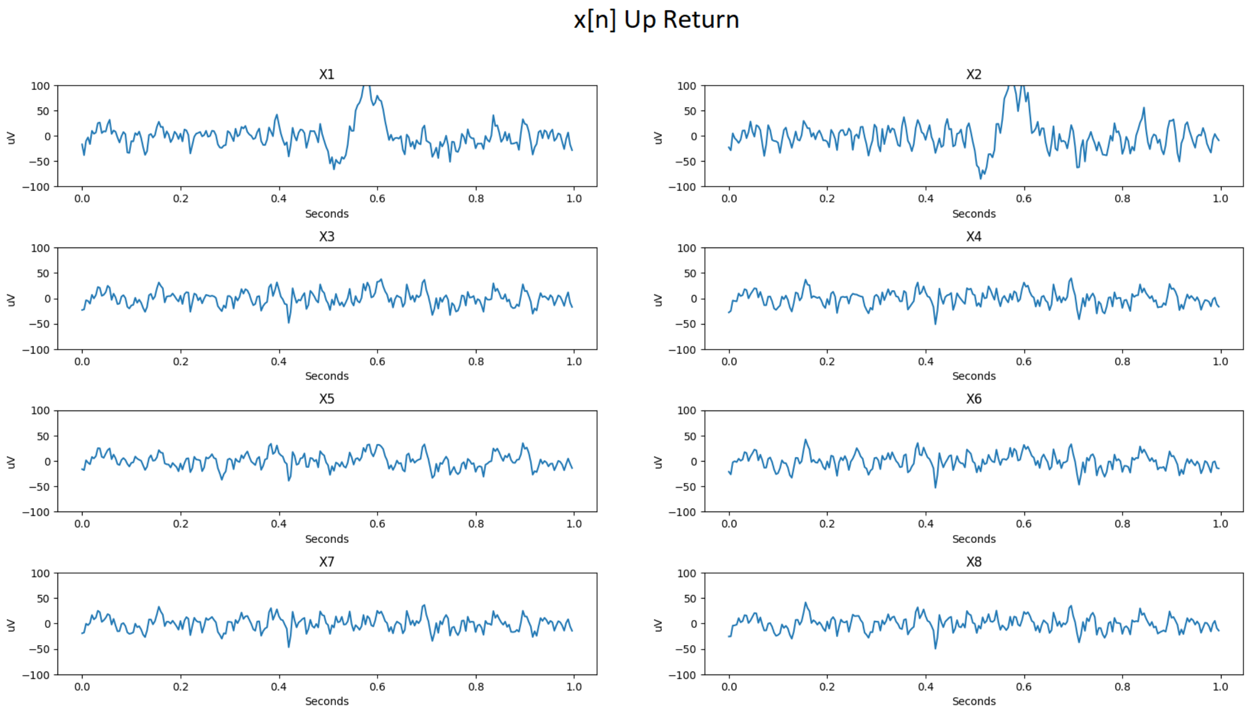


Figure B.3: Back to center from up at 0.6 seconds

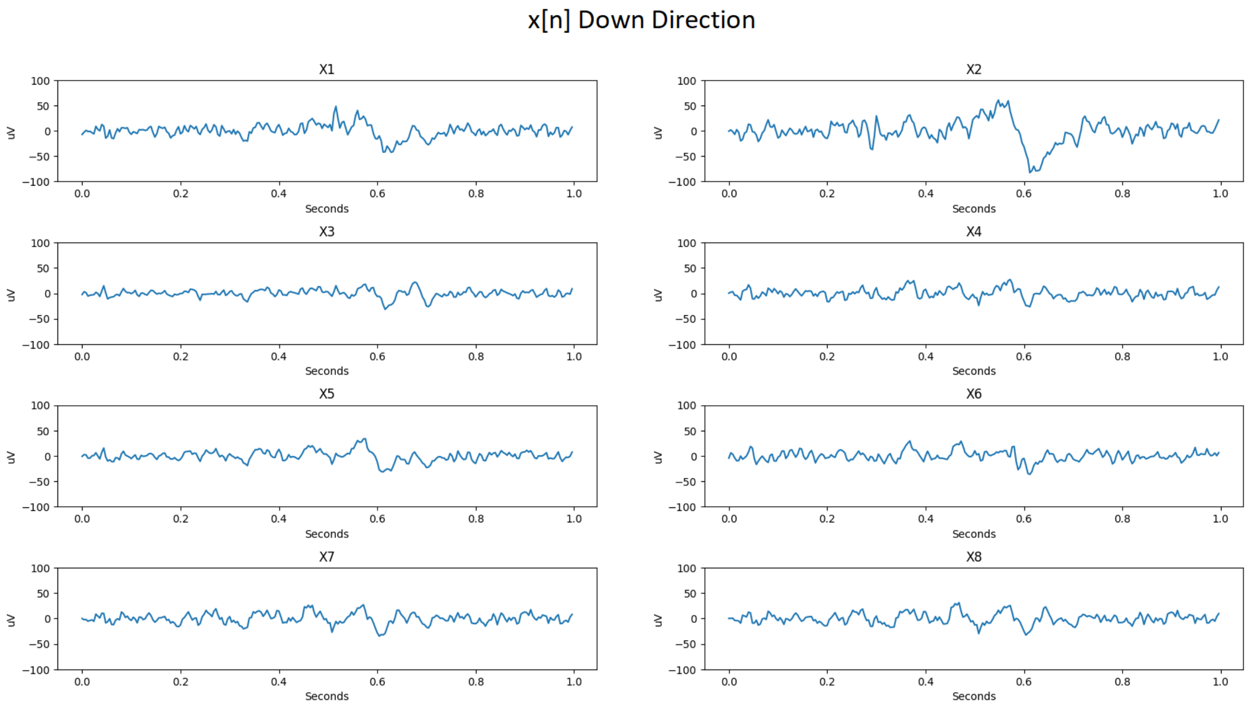


Figure B.4: Down from center at 0.6 seconds.

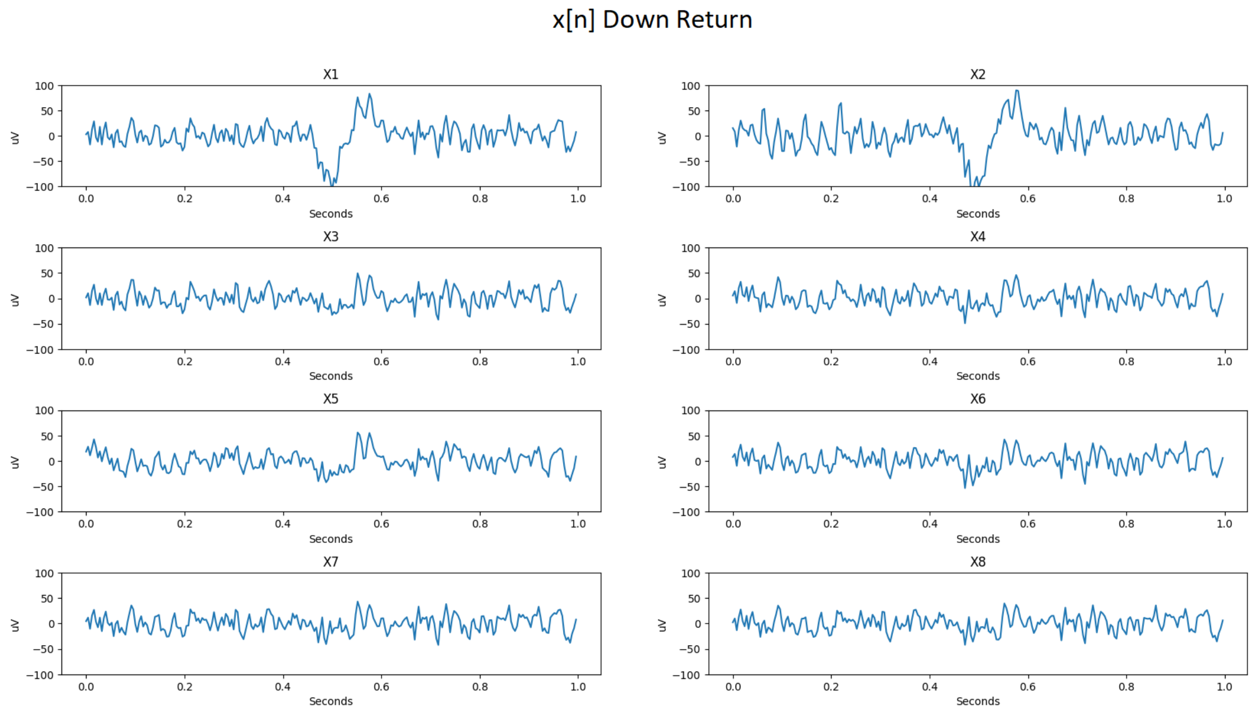


Figure B.5: Back to center from down at 0.5 seconds.

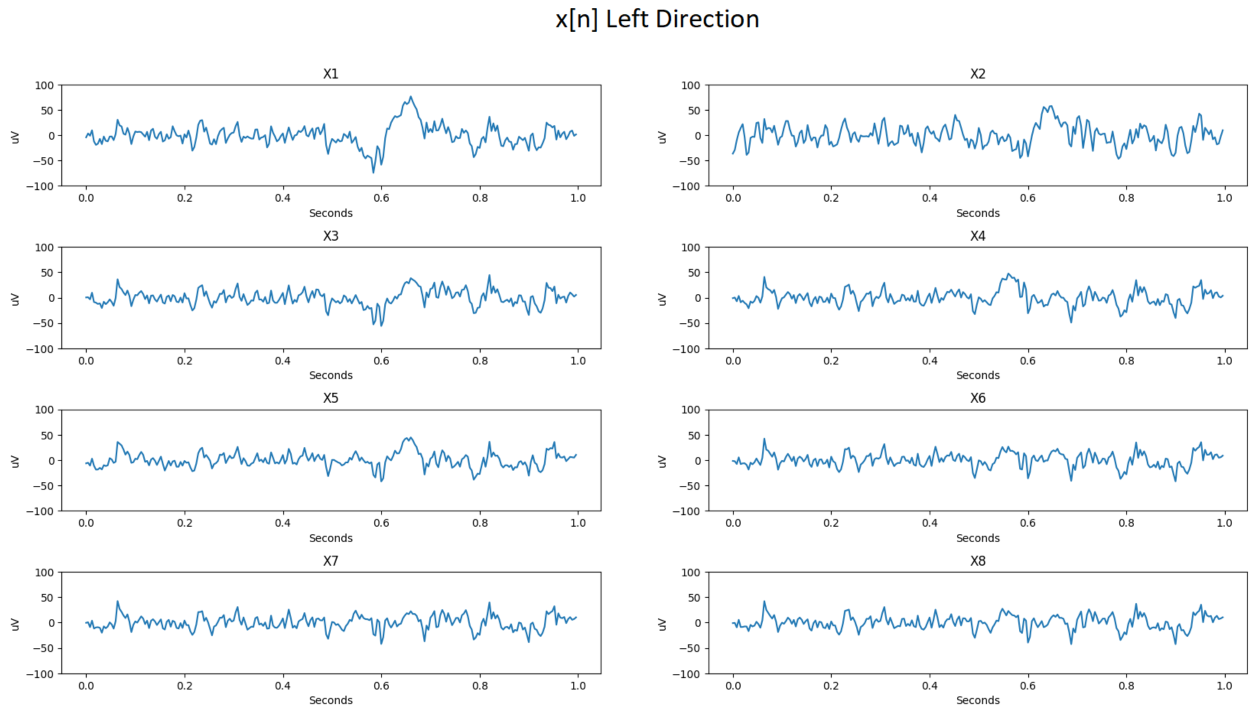


Figure B.6: Left from center at 0.6 seconds.

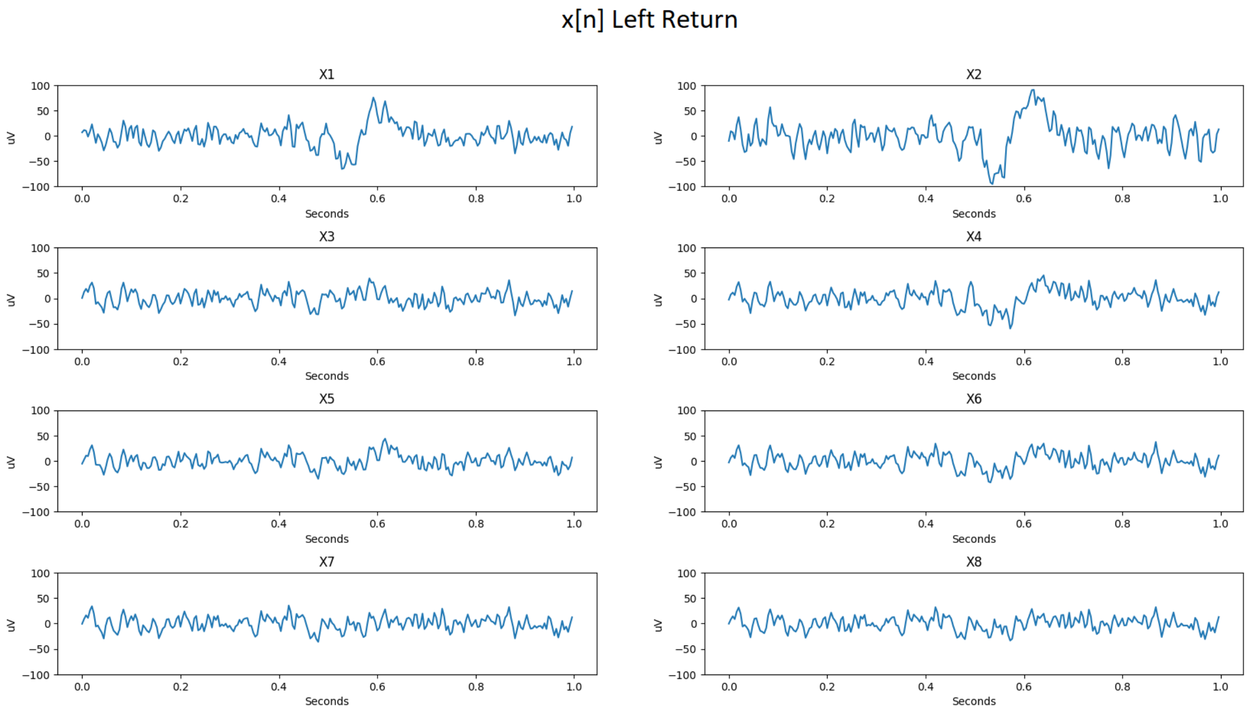


Figure B.7: Back to center from left at 0.6 seconds.

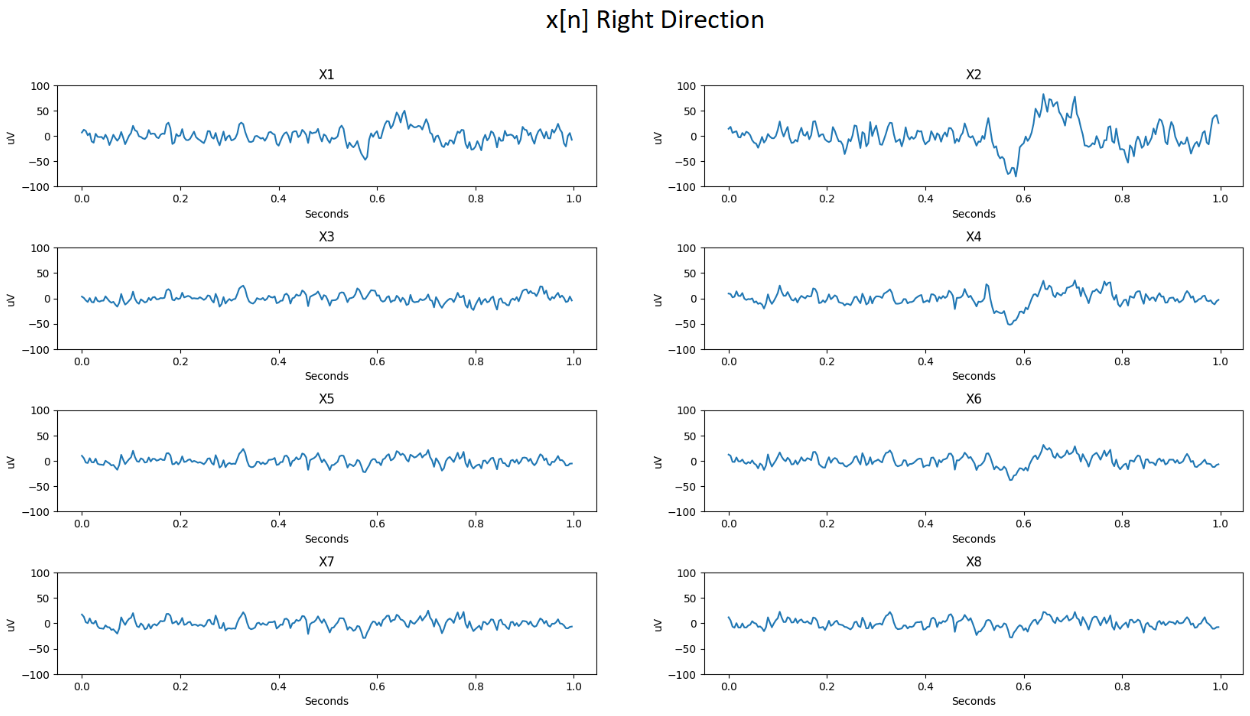


Figure B.8: Right from center at 0.6 seconds.

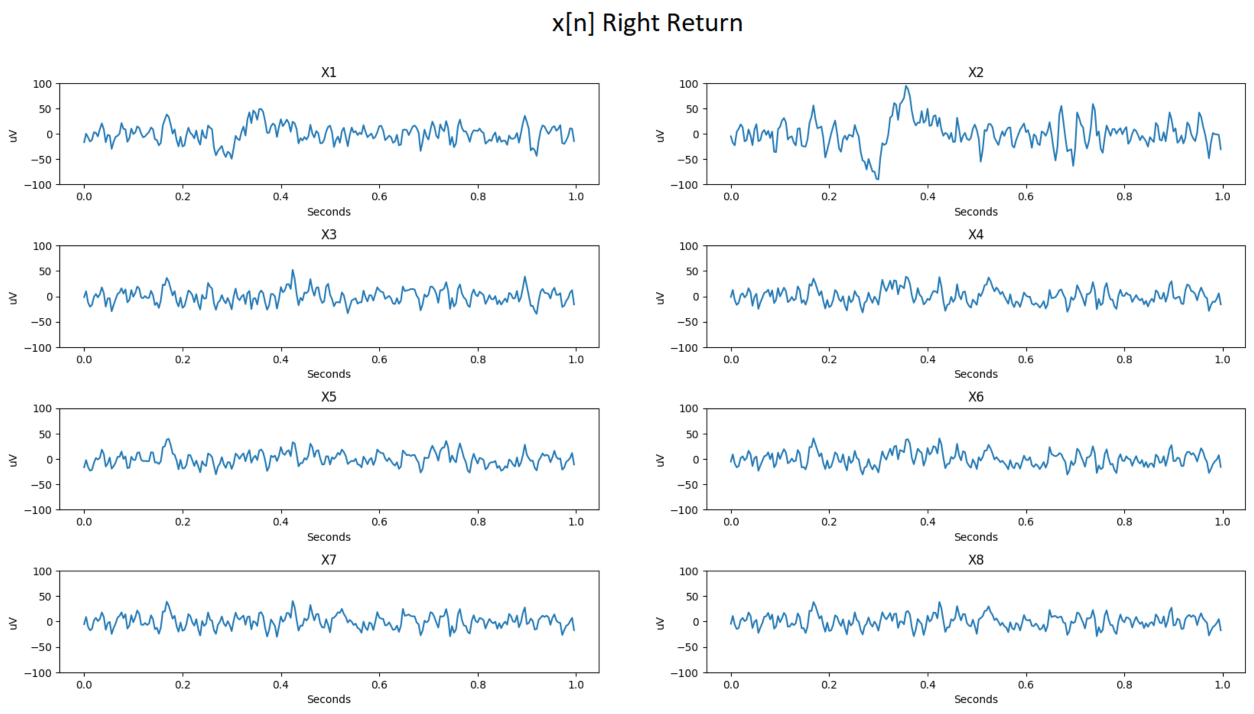


Figure B.9: Back to center from right at 0.3 seconds.

Appendix C

Calculation procedures for features

Several features have been presented in Section 4.1.2. However, the presentation of the features has avoided going into detail around the calculation of the features due to the complexity of some of them and that they are not being used for anything further in the thesis. This Appendix goes into detail about how several of the proposed features are defined and how they are calculated. This is not definitions from own work but is simply a description of the implementation of features found in other literature. In some cases, the text is directly copied from the original literature.

C.1 Power Spectral Entropy

The computational procedure for calculating the Power Spectral Entropy (PSE), as defined in [67] follows:

To a time series $[x_1, x_2, \ldots, x_N]$, denote its Fast Fourier Transform (FFT) result as $[X_1, X_2, \ldots, X_N]$. A continuous frequency band from f_{low} to f_{up} is sliced into K bins, which can be of equal width or not. Boundaries of bins are specified by a vector band $= [f_1, f_2, \ldots, f_K]$, such that the lower and upper frequencies of the i-th bin are f_i and f_{i+1}, respectively. Commonly used unequal bins are EEG/MEG rhythms, which are, $\delta(0.5-4\text{ Hz})$, $\theta(4-7\text{ Hz})$, $\alpha(8-12\text{ Hz})$, $\beta(12-30\text{ Hz})$, and $\gamma(30-100\text{ Hz})$. For these bins, we have band $= [0.5, 4, 7, 12, 30, 100]$. The Power Spectral Intensity (PSI) [12] of the k-th bin is defined in equation C.1

$$\text{PSI}_k = \sum_{i=\lfloor N(f_k/f_s)\rfloor}^{\lfloor N(f_{k+1}/f_s)\rfloor} |X_i| \quad k = 1, 2, \ldots, K-1, \tag{C.1}$$

where f_s is the sampling rate, and N is the series length. Relative Intensity Ratio (RIR) is defined on top of PSI in equation C.2

$$\text{RIR}_j = \frac{\text{PSI}_j}{\sum_{k=1}^{K-1}\text{PSI}_k}, \quad j = 1, 2, \ldots, K-1 \tag{C.2}$$

The Power Spectral Entropy (PSE) is defined in equation C.3

$$\text{PSE} = -\tfrac{1}{\log(K)} \textstyle\sum_{i=1}^{K} \text{RIR}_i \log \text{RIR}_i \tag{C.3}$$

C.2 Higuchi Fractal Dimension

The Higuchi Fractal Dimension (HFD) algorithm [64] approximates the mean length of curve segments of length k. The algorithm as defined in [67] follows:

Given a time series observations taken at regular interval: $X(1), X(2), ..., X(N)$ we first construct a new time series X_k^m in Equation C.4.

$$X_k^m : X(m), X(m+k), X(m+2k), \ldots,\ X\left(m + \left(\frac{N-m}{k}\right)k\right) \tag{C.4}$$

Where $m = 1, 2, \dots, k$.

We then calculate the length of the curve X_k^m in Equation C.5

$$L_m(k) = \left\{ \left(\sum_{i=1}^{\left[\frac{N-m}{k}\right]} |X(m+ik) - X(m+(i-1)k)| \right) \frac{N-1}{\left[\frac{N-m}{k}\right]k} \right\} \Big/ k \tag{C.5}$$

C.3 Pearson Correlation Coefficients

Equation C.6 shows how the correlation is calculated as defined in [131]:

$$r = \frac{\sum_{i=1}^{l_{\mathrm{w,c}}} (x_i - \bar{x})(x_i - \bar{x})}{\sqrt{\sum_{i=1}^{l_{\mathrm{w,c}}} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{l_{\mathrm{w,c}}} (x_i - \bar{x})^2}} \tag{C.6}$$

C.4 Theta-Beta ratio

For notation purposes, the FFT, of $x_m[n]$ specific to the Theta and Beta bands (defined in table 2.1) are noted as $X_\theta[n]$ and $X_\beta[n]$. The TBR is defined by Equation C.7 [41].

$$TBR = \frac{\frac{1}{l_{w,c}} \sum_{n=1}^{l_{w,c}} X_{n\theta}}{\frac{1}{l_{w,c}} \sum_{n=1}^{l_{w,c}} X_{n\beta}}, \tag{C.7}$$

where n denotes the sample index.

C.5 Petrosian Fractal Dimension

The Petrosian Fractal Dimension (PFD) is defined in Equation C.8 [67].

$$\mathrm{PFD} = \frac{\log_{10} N}{\log_{10} N + \log_{10}(N/(N + 0.4 N_\delta))}, \tag{C.8}$$

where N is the series length, and N_δ is the number of sign changes in the signal derivative

C.6 Covariance

To calculate the covariance for $x_j \wedge x_k$ the following approximation are used [69]:

$$C \approx \frac{1}{P} \sum_{i=0}^{N-1} (\mathbf{x}_{i,j} - \bar{\mathbf{x}})(\mathbf{x}_{i,k} - \bar{\mathbf{x}})^H, \tag{C.9}$$

where x_i is an observation of $\mathbf{X}$ (as a column-vector), N is the number of observations made and $P = N - 1$.

Appendix D

Offline testing Confusion Matrices for Subject 2

Subject 1 was used as a baseline when presenting the different models and their performance in the offline results section 5.5. Subject 2 also performed the same tests. The following Appendix shows the Confusion Matrices and Classification Reports for Subject 2.

D.1 SVM with RBF kernel and samples as features for Subject 2

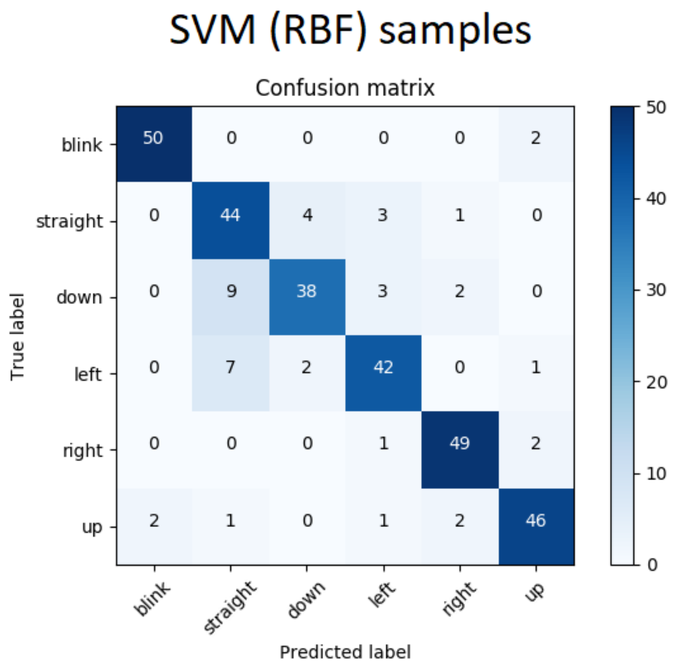


Figure D.1: Confusion Matrix for SVM with RBF-kernel and samples as features on the test set for Subject 2. An ideal model would have 52 true positives for each class

The accuracy on the test set was shown to be $A = 82.7\%$, where the Classification Report in Table D.1 shows different scores for the respective classes.

Table D.1: Classification Report Subject 2

Class	Precision	Recall	f1-score	support
Blink	1.00	0.96	0.98	52
Down	0.75	0.79	0.77	52
Left	0.73	0.67	0.70	52
Straight	0.75	0.83	0.79	52
Right	0.86	0.81	0.83	52
Up	0.89	0.90	0.90	52
avg/total	0.83	0.83	0.83	312

D.2 SVM with RBF kernel and feature selection Subject 2

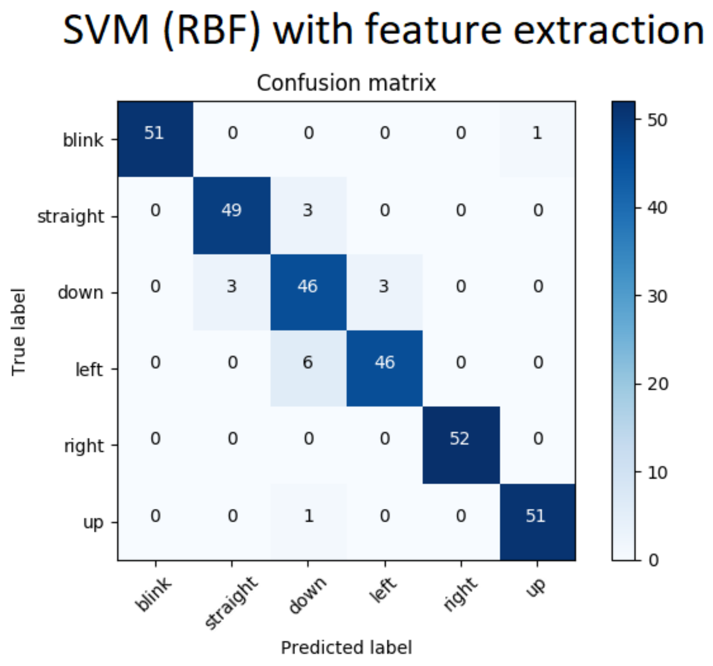


Figure D.2: Confusion Matrix for SVM with RBF-kernel on the test set for Subject 2 with extracted features. An ideal model would have 52 true positives for each class

The accuracy on the test set was shown to be $A = 94.5\%$, where the Classification Report in Table D.2 shows different scores for the respective classes.

Table D.2: Classification Report Subject 2

Class	Precision	Recall	f1-score	support
Blink	1.00	0.98	0.99	52
Down	0.82	0.88	0.85	52
Left	0.94	0.88	0.91	52
Straight	0.94	0.94	0.94	52
Right	1.00	1.00	1.00	52
Up	0.98	0.98	0.98	52
avg/total	0.95	0.95	0.95	312

D.3 SVM with linear kernel Subject 2

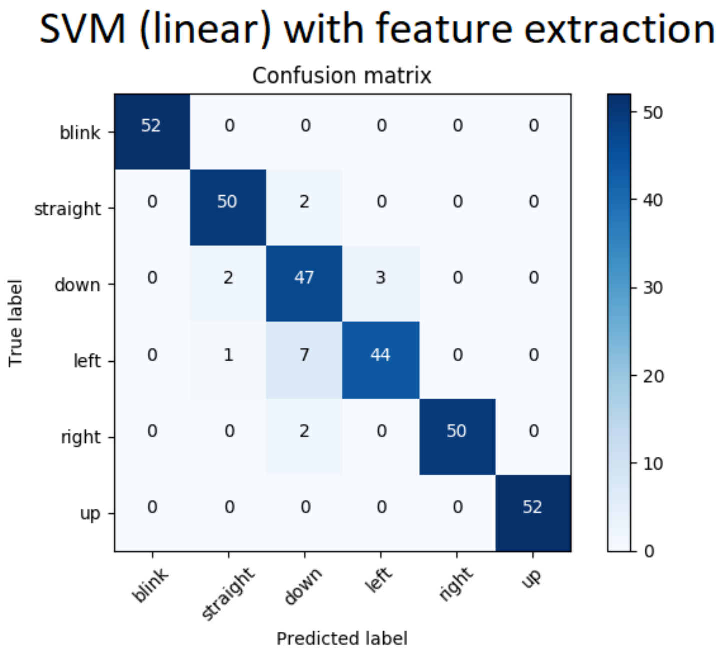


Figure D.3: Confusion Matrix for SVM with linear-kernel on the test set for Subject 2 with extracted features. An ideal model would have 52 true positives for each class

The accuracy on the test set was shown to be 94.5%, where the Classification Report in Table D.3 shows different scores for the respective classes.

Table D.3: Classification Report Subject 2

Class	Precision	Recall	f1-score	support
Blink	1.00	1.00	1.00	52
Down	0.81	0.90	0.85	52
Left	0.94	0.85	0.89	52
Straight	0.94	0.96	0.95	52
Right	1.00	0.96	0.98	52
Up	1.00	1.00	1.00	52
avg/total	0.95	0.95	0.95	312

D.4 LinearSVC Subject 2

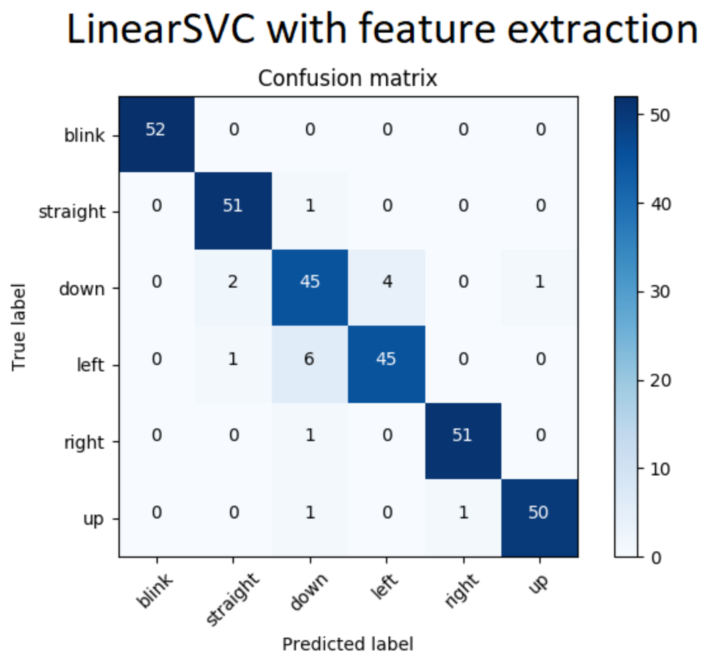


Figure D.4: Confusion Matrix for LinearSVC on the test set for Subject 2 with extracted features. An ideal model would have 52 true positives for each class

The accuracy on the test set was shown to be 94.2%, where the Classification Report in Table D.4 shows different scores for the respective classes.

Table D.4: Classification Report Subject 2

Class	Precision	Recall	f1-score	support
Blink	1.00	1.00	1.00	52
Down	0.83	0.87	0.85	52
Left	0.92	0.87	0.89	52
Straight	0.94	0.98	0.96	52
Right	0.98	0.98	0.98	52
Up	0.98	0.96	0.97	52
avg/total	0.94	0.94	0.94	312

Appendix E

Offline testing with Random Forest and K-Nearest Neighbors

Several different algorithms were tested offline in 5.5. Some was shown to have a larger focus than others due to their results. This Appendix shows the Confusion Matrices and Classification Reports of the Random Forest and K-nearest neighbor classifiers in 5.5.

E.1 Random Forest

E.1.1 Subject 1

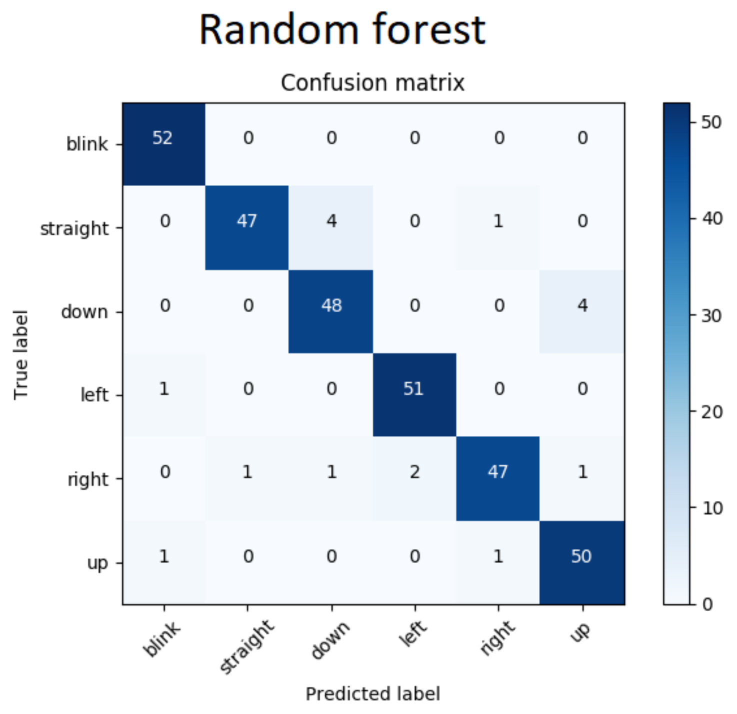


Figure E.1: Confusion Matrix for Random forest on the test set for Subject 1 with extracted features. An ideal model would have 52 true positives for each class

Table E.1: Classification Report Subject 1

Class	Precision	Recall	f1-score	support
Blink	0.96	1.00	0.98	52
Down	0.91	0.92	0.91	52
Left	0.96	0.98	0.97	52
Straight	0.98	0.90	0.94	52
Right	0.96	0.90	0.93	52
Up	0.91	0.96	0.93	52
avg/total	0.95	0.95	0.95	312

E.1.2 Subject 2

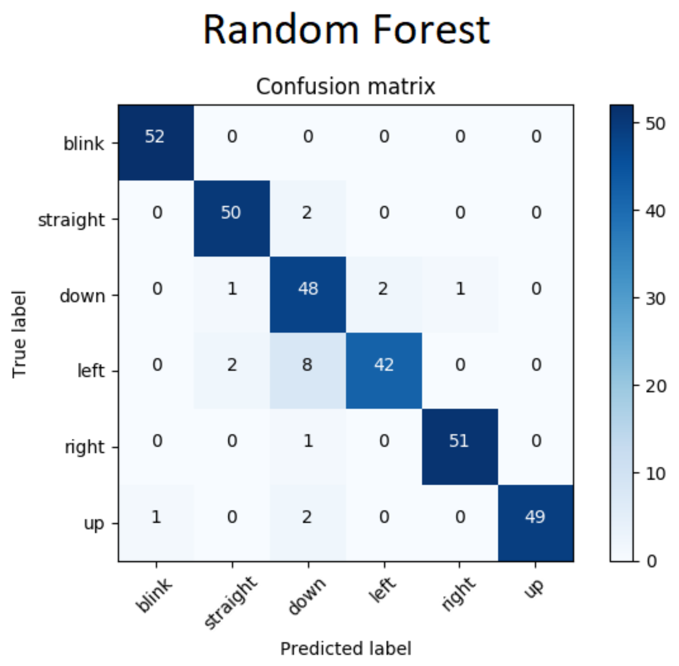


Figure E.2: Confusion Matrix for Random forest on the test set for Subject 2 with extracted features. An ideal model would have 52 true positives for each class

The accuracy on the test set was shown to be 93.5%, where the Classification Report in Table D.2 shows different scores for the respective classes.

Table E.2: Classification Report Subject 2

Class	Precision	Recall	f1-score	support
Blink	0.98	1.00	0.99	52
Down	0.79	0.92	0.85	52
Left	0.95	0.81	0.88	52
Straight	0.94	0.96	0.95	52
Right	0.98	0.98	0.98	52
Up	1.00	0.94	0.97	52
avg/total	0.94	0.94	0.94	312

E.2 K-Nearest Neighbors

E.2.1 Subject 1

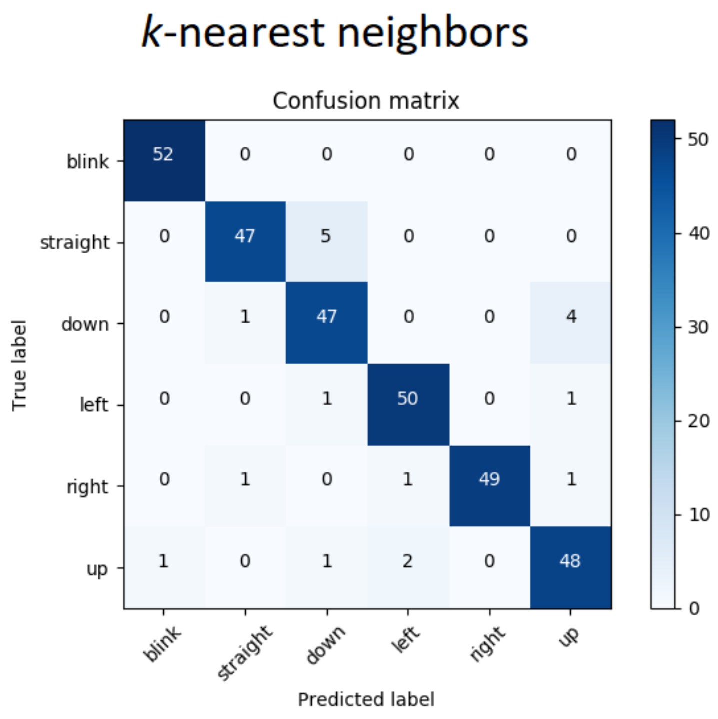


Figure E.3: Confusion Matrix for *k*-nearest neighbors on the test set for Subject 1 with extracted features. An ideal model would have 52 true positives for each class

Table E.3: Classification Report Subject 1

Class	Precision	Recall	f1-score	support
Blink	0.98	1.00	0.99	52
Down	0.87	0.90	0.89	52
Left	0.94	0.96	0.95	52
Straight	0.96	0.90	0.93	52
Right	1.00	0.94	0.97	52
Up	0.89	0.92	0.91	52
avg/total	0.95	0.95	0.95	312

E.2.2 Subject 2

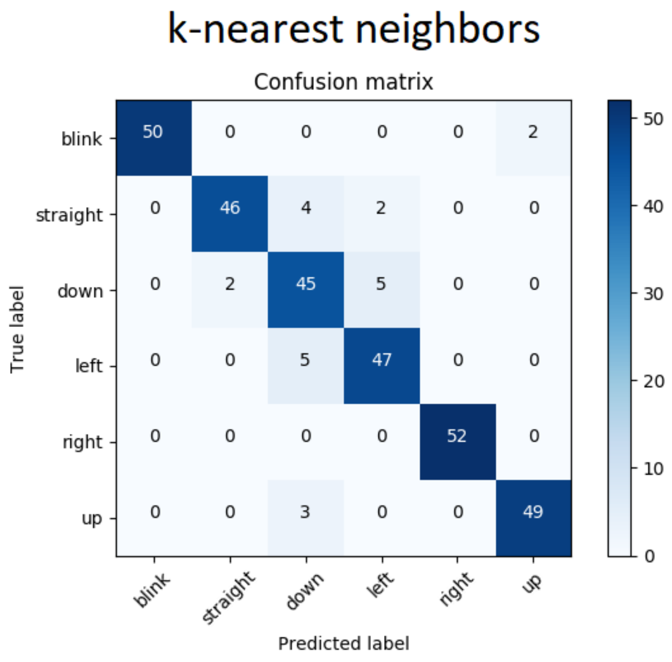


Figure E.4: Confusion Matrix for K-nearest neighbors on the test set for Subject 2 with extracted features. An ideal model would have 52 true positives for each class

The accuracy on the test set was shown to be 92.6%, where the Classification Report in Table E.4 shows different scores for the respective classes.

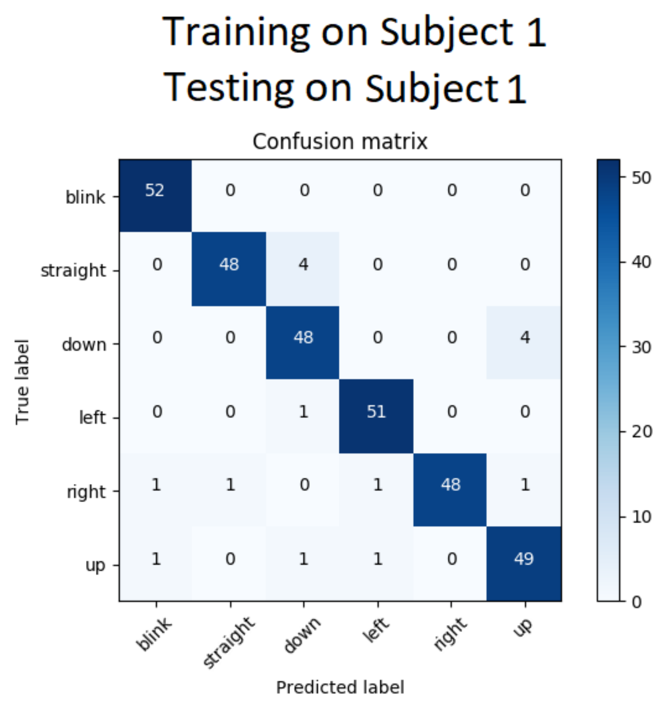
Table E.4: Classification Report Subject 2

Class	Precision	Recall	f1-score	support
Blink	1.00	0.96	0.98	52
Down	0.79	0.87	0.83	52
Left	0.87	0.90	0.89	52
Straight	0.96	0.98	0.92	52
Right	1.00	1.00	1.00	52
Up	0.96	0.94	0.95	52
avg/total	0.93	0.93	0.93	312

Appendix F

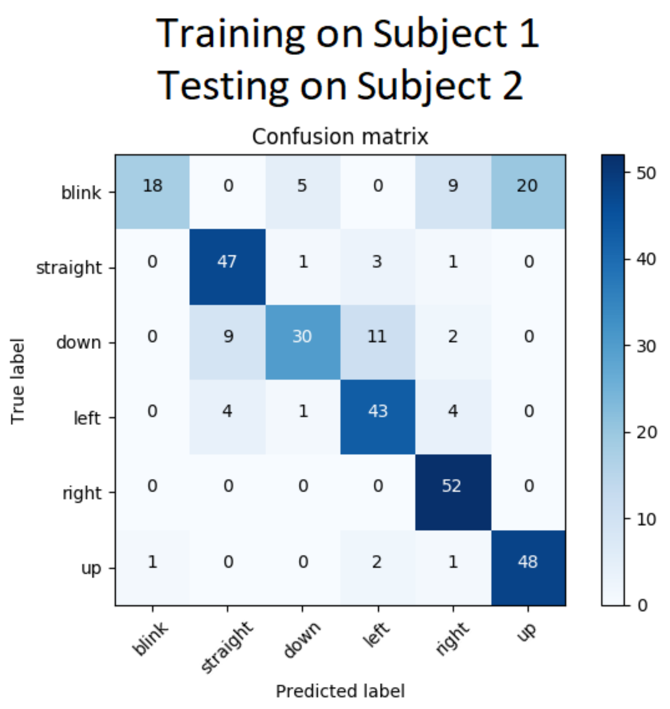
Subject comparison

The following appendix contains Confusion Matrices from the cross-subject comparison when testing the SVM model with RBF-based Kernel in an offline setting as presented in 5.5.



	blink	straight	down	left	right	up
blink	52	0	0	0	0	0
straight	0	48	4	0	0	0
down	0	0	48	0	0	4
left	0	0	1	51	0	0
right	1	1	0	1	48	1
up	1	0	1	1	0	49

Figure F.1: Showing the Confusion Matrix when training on Subject 1 and testing on Subject 1. An ideal model would have 52 true positives for each class



	blink	straight	down	left	right	up
blink	18	0	5	0	9	20
straight	0	47	1	3	1	0
down	0	9	30	11	2	0
left	0	4	1	43	4	0
right	0	0	0	0	52	0
up	1	0	0	2	1	48

Figure F.2: Showing the Confusion Matrix when training on Subject 1 and testing on Subject 2. An ideal model would have 52 true positives for each class

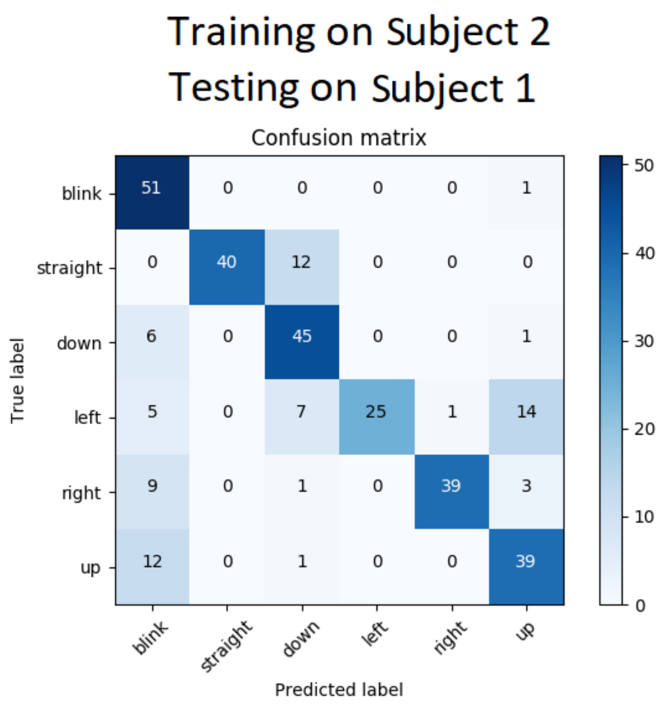


Figure F.3: Showing the Confusion Matrix when training on Subject 2 and testing on Subject 1. An ideal model would have 52 true positives for each class

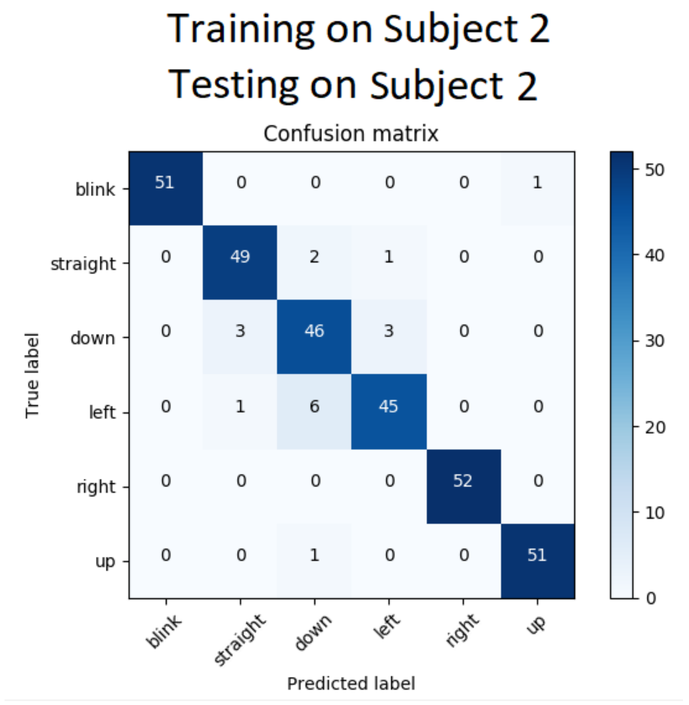


Figure F.4: Showing the Confusion Matrix when training on Subject 2 and testing on Subject 2. An ideal model would have 52 true positives for each class

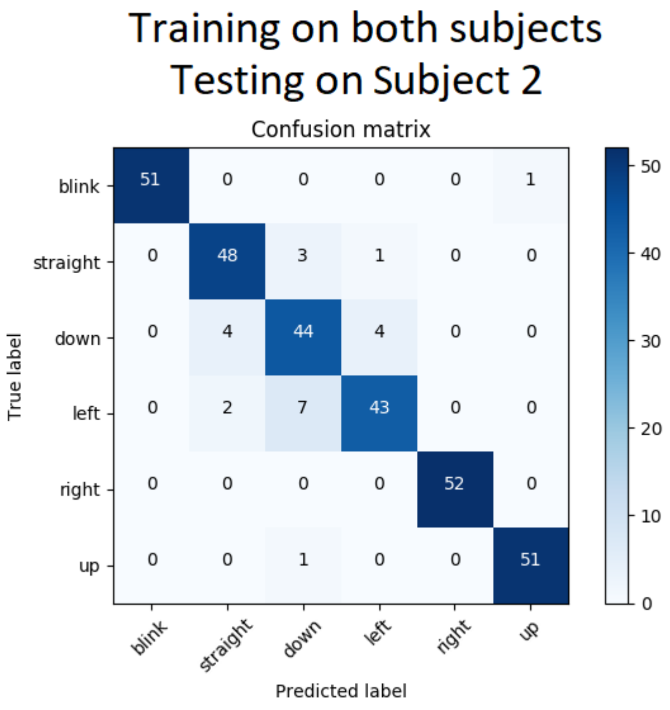


Figure F.5: Showing the Confusion Matrix when training on both subjects and testing on Subject 2. An ideal model would have 104 true positives for each class

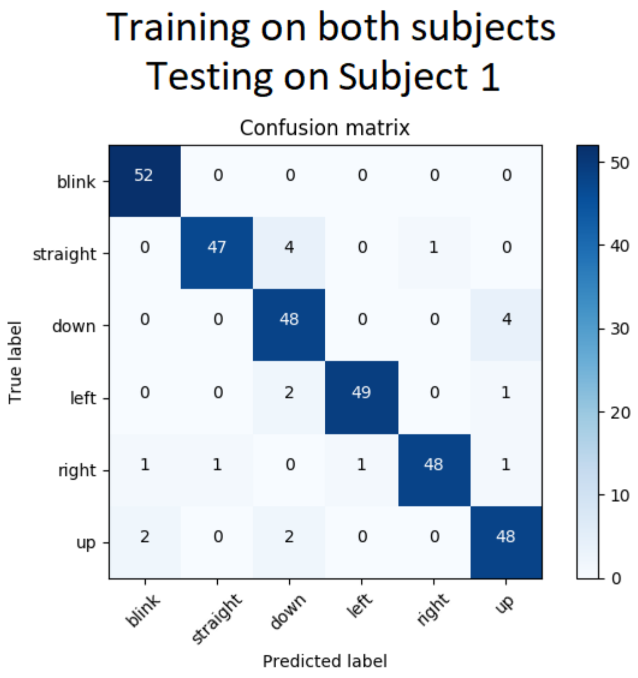


Figure F.6: Showing the Confusion Matrix when training on both subjects and testing on Subject 1. An ideal model would have 104 true positives for each class

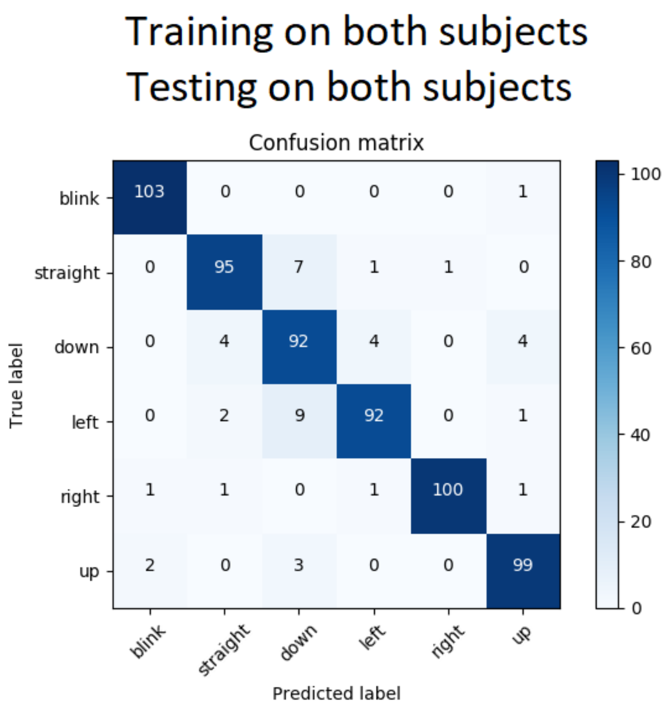


Figure F.7: Showing the Confusion Matrix when training on both subjects and testing on both subjects. An ideal model would have 104 true positives for each class

Appendix G

Software implementation

There has been written an extensive amount of code for this project. The code size is over 6k lines, so it is both too big to be put in Appendix and it also serves no purpose for the report. The software implementation can be found in the Github repository `https://github.com/kristiankrohn/Masterproject`. This Appendix shows the dependencies needed to utilize the software created in this work.

In addition to direct use of open source modules, there are also some of the libraries that have been modified and put in the repository. These files are emd.py, hht.py, mttkinter.py and pyeeg.py. The rest are used as downloaded.

G.1 Dependencies

The software heavily depends on native Python 2.7 libraries, but also on some external libraries:

- Python 2.7.14
- Numpy 1.13.1
- Scipy 0.19.1
- PyQtGraph 0.10.0
- PyQt4
- scikit-learn 0.19.1
- open_bci_v3
- keyboard 0.11.0
- PyHht 0.1.0
- tracestack 0.2.4
- ps_drone
- pandas 0.21.0
- seaborn 0.8.1
- matplotlib 2.0.
- pyeeg 0.02 r1
- dill 0.2.7.1

- mttkinter 0.5.0
- netifaces 0.10.6
- psutil 5.4.5
- pypiwin32 223
- pyserial 3.4

To run the program, you will also need an external folder structure to hold the datasets, plot exports and logs, which can be seen in figure G.1.

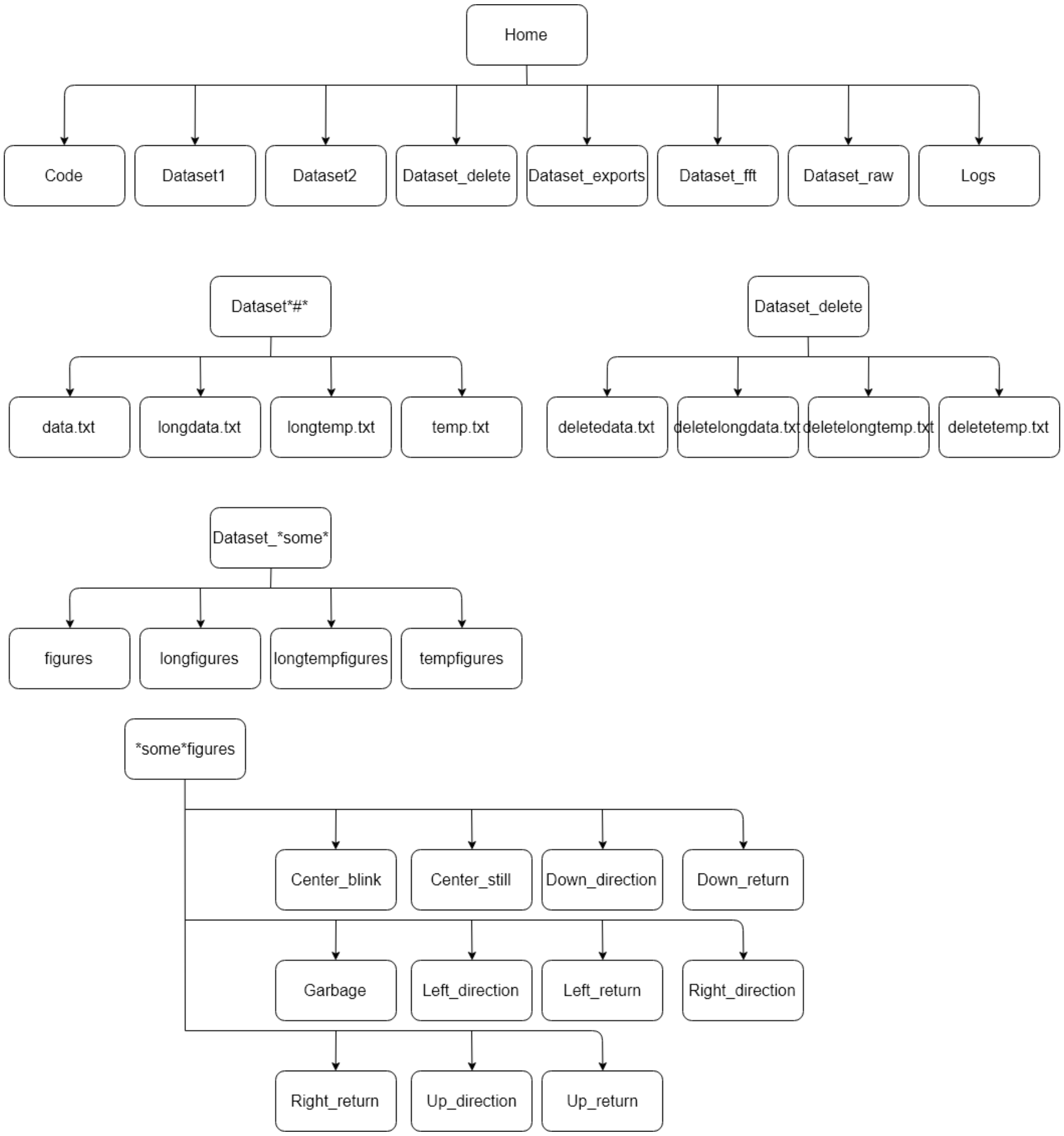


Figure G.1

The top directory is on top, the rest are subdirectories. Names with * are unspecified, meaning that Dataset_*some* is for Dataset_exports, Dataset_fft and Dataset_raw unless specified otherwise.

G.2 Basic usage and workflow

To start the system just run the main.py file. The system has a terminal interface where you can choose what to do. Take a look in the file to find out what they are, the "help" option does not include all options. To start with the training gui you need to run the main.py with gui as an input parameter. start to collect data from the helmet with "start" command. When you have made some data, export the plots with "exporttempplots" command. To delete datapoints you type in a list of indices found on each plot in the corresponding file in Dataset_delete and then run the "deleteappendedtemp" command. After that you can append the new datapoints to the dataset with the "savedata" command. For training the system it is easier to look in the code, but to make a classifier for online prediction you need to run the createPredictor function in predict.py by writing in the settings and running the predict.py file. To start the online system, you first need to start to collect data from the helmet, then start the TTS system with "speak" command and then start the drone with "drone" or the online verification controller with "online".

G.3 Tips

If you are going to use this system, the main suggestion is that you redesign the dataset storage, as to store large amounts of floats as text is a very inefficient storage solution, but it was nice for debugging the system and was never changed after that. Good luck!

Appendix H

Controller

A controller was needed to use the predictions from the classifier to translate into drone commands. The following Appendix shows the code-implementation of the controller in Python.

Listing H.1: State-machine implementation

```
def stateMachine(blinks, opposite, otherkey, pressedKey, keypress,
  previousPrediction, prevPreviousPrediction, gotfive, gotother,
  gotopposite, lastTime, drone=None):
  #global speak
  now = datetime.now()

  if (now - lastTime) > timedelta(seconds=3):
    '''
    if blinks > 0: #Use this for decrement every interval
      blinks = blinks - 1
    else:
      blinks = 0
    '''
    blinks = 0 #Use this for resetting blinks every interval
    lastTime = now

  try:
    with glb.predictionslock:
      prediction = glb.predictionsQueue.get(block=False, timeout=1)
      print(prediction)
  except:
    prediction = None

  if prediction != None:

    #State transition from P0 to P1

    if prediction == 0:
      if previousPrediction == 0:
        if prevPreviousPrediction != 0 and not keypress:
            blinks += 1
            print("Blinks:␣%d" %blinks)
            speak.Speak(str(blinks) + "blinks")
            lastTime = datetime.now()

#State transition from S0 to S1
if not keypress and prediction == previousPrediction:

    #if prediction in [2,4,6,8]:
    if prediction in [4,6,8]: #without backwards
        keypress = True
        pressedKey = prediction

        if prediction == 8:
            if drone != None:
                drone.moveForward(0.1)
            print("Move␣forwards")
            speak.Speak("Forwards")
        elif prediction == 2:
            if drone != None:
                drone.moveBackward()
            print("Move␣backwards")
            speak.Speak("Backwards")
        elif prediction == 4:
            print("Turn␣left")
            speak.Speak("Left")
            if drone != None:
                drone.turnLeft(1)
                tme.sleep(0.2)
            drone.hover()
          #print("Finished turning left")
        elif prediction == 6:
          print("Turn␣right")
          speak.Speak("Right")
          if drone != None:
            drone.turnRight(1)
            tme.sleep(0.2)
            drone.hover()
          #print("Finished turning right")

    if keypress:
      #State transition from S1 to S2
      if prediction == opposite[pressedKey]:
        if gotopposite == 0:
          print("Hover")
          #speak.Speak("Hover")
          if drone != None:
            drone.hover()
        gotopposite += 1

      elif prediction in otherkey[pressedKey]:
        if gotother == 0:
          print("Hover")
          #speak.Speak("Hover")
          if drone != None:
            drone.hover()
        elif gotother == 2:
          #Do the other movement
          pass
```

```
                gotother += 1

            #State transistion from S2 to S0
            if prediction == 5:
                if (gotopposite == 2 or gotother == 2):
                    gotother = 0
                    gotopposite = 0
                    keypress = False
                    speak.Speak("Finished")
                    print("Ready for new prediction, other/opposite exit")

                elif pressedKey in [4,6]:
                    if ((previousPrediction == 5) and
                        (prevPreviousPrediction == 5)):
                        #speak.Speak("Hover")
                        print("Hover")
                        if drone != None:
                            drone.hover()
                        gotother = 0
                        gotopposite = 0
                        keypress = False
                        speak.Speak("Finished")
                        print("Ready for new prediction, five exit")

        prevPreviousPrediction = previousPrediction
        previousPrediction = prediction

    return blinks, opposite, otherkey, pressedKey, keypress,\
        previousPrediction, prevPreviousPrediction, gotfive,\
        gotother, gotopposite, lastTime
```

Appendix I

Feature vectors for online testing

When creating models for online testing, RFECV was used to choose features for the different models as presented in section 6.5. Only the performance metrics achieved from the model was presented in 6.5. The following Appendix shows the feature vectors associated with the different presented models.

I.1 Testing with down as a separate command

I.1.1 Subject 1 SVM with RBF

Table I.1: Feature indices, names and channels

Index	Name	Channel
0	Higuchi Fractal Dimension	1
1	Higuchi Fractal Dimension	4
2	Minimum Value Difference	1 and 3
3	Maximum Value Difference	1 and 3
4	Spectral Entropy	1
6	Pearson Coefficient	3 and 4
7	Pearson Coefficient	3 and 4
9	Pearson Coefficient	1 and 4
10	Covariance	3 and 4
11	Covariance	3 and 4
12	Covariance	1 and 4
13	Covariance	1 and 4
14	Standard Deviation	1
15	Standard Deviation	4
16	Slope	1
18	$\theta\beta$ ratio	1
22	Peak-to-Peak	1
24	Minimum Value	1
25	Maximum Value	1

I.1.2 Subject 2 LinearSVC with down

Table I.2: Feature indices, names and channels

Index	Name	Channel
0	Higuchi Fractal Dimension	1
1	Higuchi Fractal Dimension	4
2	Minimum Value Difference	1 and 3
3	Maximum Value Difference	1 and 3
4	Spectral Entropy	1
5	Spectral Entropy	4
6	Pearson Coefficient	3 and 4
7	Pearson Coefficient	3 and 4
8	Pearson Coefficient	1 and 4
9	Pearson Coefficient	1 and 4
10	Covariance	3 and 4
11	Covariance	3 and 4
12	Covariance	1 and 4
13	Covariance	1 and 4
14	Standard Deviation	1
15	Standard Deviation	4
16	Slope	1
17	Slope	4
18	$\theta\beta$ ratio	1
19	$\theta\beta$ ratio	4
20	Petrosian Fractal Dimension	1
21	Petrosian Fractal Dimension	4
22	Peak-to-Peak	1
23	Peak-to-Peak	4
24	Minimum Value	1
25	Maximum Value	1

I.2 Testing without down as a separate command

I.2.1 Subject 1 LinearSVC

Table I.3: Feature indices, names and channels

Index	Name	Channel
0	Higuchi Fractal Dimension	1
1	Higuchi Fractal Dimension	4
2	Minimum Value Difference	1 and 3
3	Maximum Value Difference	1 and 3
4	Spectral Entropy	1
6	Pearson Coefficient	3 and 4
7	Pearson Coefficient	3 and 4
9	Pearson Coefficient	1 and 4
10	Covariance	3 and 4
11	Covariance	3 and 4
12	Covariance	1 and 4
13	Covariance	1 and 4
14	Standard Deviation	1
15	Standard Deviation	4
16	Slope	1
18	$\theta\beta$ ratio	1
22	Peak-to-Peak	1
24	Minimum Value	1
25	Maximum Value	1

I.2.2 Subject 2 LinearSVC

Same as with the model for Subject 2 based on LinearSVC with down in section I.1.2

I.2.3 Subject 1 SVM with RBF

Same as with Subject 1 based on SVM with RBF with down in section I.1.1

I.2.4 Subject 2 SVM with RBF

Table I.4: Feature indices, names and channels

Index	Name	Channel
1	Higuchi Fractal Dimension	4
2	Minimum Value Difference	1 and 3
3	Maximum Value Difference	1 and 3
4	Spectral Entropy	1
7	Pearson Coefficient	3 and 4
8	Pearson Coefficient	1 and 4
10	Covariance	3 and 4
11	Covariance	3 and 4
12	Covariance	1 and 4
13	Covariance	1 and 4
14	Standard Deviation	1
15	Standard Deviation	4
16	Slope	1
18	$\theta\beta$ ratio	1
19	$\theta\beta$ ratio	4
23	Peak-to-Peak	4
24	Minimum Value	1
25	Maximum Value	1

I.2.5 Subject 1 Unified SVM with RBF

Table I.5: Feature indices, names and channels

Index	Name	Channel
0	Higuchi Fractal Dimension	1
1	Higuchi Fractal Dimension	4
2	Minimum Value Difference	1 and 3
3	Maximum Value Difference	1 and 3
4	Spectral Entropy	1
5	Spectral Entropy	4
6	Pearson Coefficient	3 and 4
7	Pearson Coefficient	3 and 4
8	Pearson Coefficient	1 and 4
9	Pearson Coefficient	1 and 4
10	Covariance	3 and 4
11	Covariance	3 and 4
12	Covariance	1 and 4
13	Covariance	1 and 4
14	Standard Deviation	1
15	Standard Deviation	4
16	Slope	1
18	$\theta\beta$ ratio	1
19	$\theta\beta$ ratio	4
22	Peak-to-Peak	1
23	Peak-to-Peak	4
24	Minimum Value	1
25	Maximum Value	1

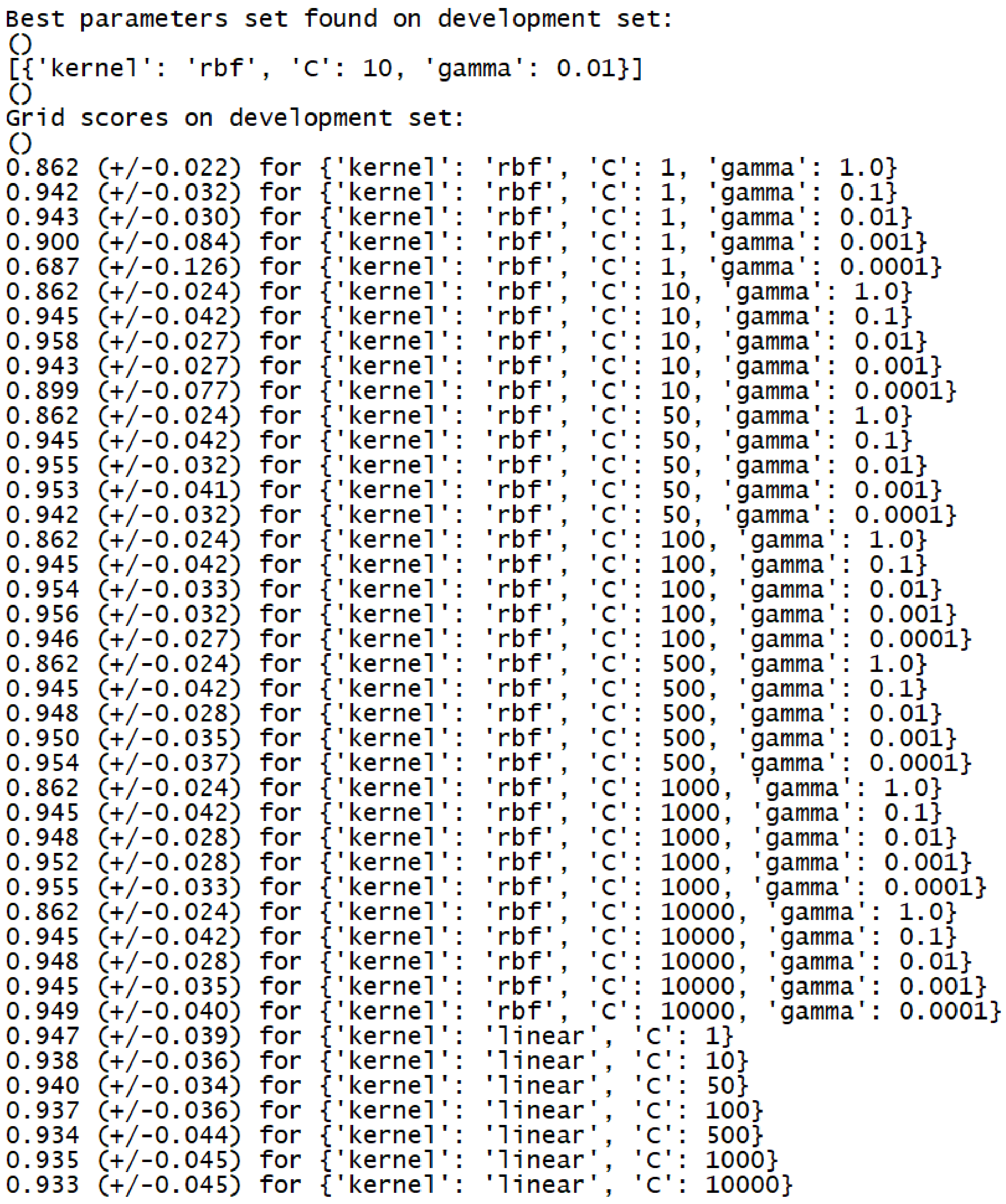
I.2.6 Subject 2 Unified SVM with RBF

Same as for Subject 1 with Unified SVM with RBF.

Appendix J

Grid search

To find the combination of hyperparameters that gives the best classifier performance a grid search was used. The grid search was provided by Scikit learn. This Appendix shows the output of the grid search, when testing hyperparameters as shown in Table 4.4.



```
Best parameters set found on development set:
()
[{'kernel': 'rbf', 'C': 10, 'gamma': 0.01}]
()
Grid scores on development set:
()
0.862 (+/-0.022) for {'kernel': 'rbf', 'C': 1, 'gamma': 1.0}
0.942 (+/-0.032) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.1}
0.943 (+/-0.030) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.01}
0.900 (+/-0.084) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.687 (+/-0.126) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.862 (+/-0.024) for {'kernel': 'rbf', 'C': 10, 'gamma': 1.0}
0.945 (+/-0.042) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.1}
0.958 (+/-0.027) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.01}
0.943 (+/-0.027) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.899 (+/-0.077) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.862 (+/-0.024) for {'kernel': 'rbf', 'C': 50, 'gamma': 1.0}
0.945 (+/-0.042) for {'kernel': 'rbf', 'C': 50, 'gamma': 0.1}
0.955 (+/-0.032) for {'kernel': 'rbf', 'C': 50, 'gamma': 0.01}
0.953 (+/-0.041) for {'kernel': 'rbf', 'C': 50, 'gamma': 0.001}
0.942 (+/-0.032) for {'kernel': 'rbf', 'C': 50, 'gamma': 0.0001}
0.862 (+/-0.024) for {'kernel': 'rbf', 'C': 100, 'gamma': 1.0}
0.945 (+/-0.042) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.1}
0.954 (+/-0.033) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.01}
0.956 (+/-0.032) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.946 (+/-0.027) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.862 (+/-0.024) for {'kernel': 'rbf', 'C': 500, 'gamma': 1.0}
0.945 (+/-0.042) for {'kernel': 'rbf', 'C': 500, 'gamma': 0.1}
0.948 (+/-0.028) for {'kernel': 'rbf', 'C': 500, 'gamma': 0.01}
0.950 (+/-0.035) for {'kernel': 'rbf', 'C': 500, 'gamma': 0.001}
0.954 (+/-0.037) for {'kernel': 'rbf', 'C': 500, 'gamma': 0.0001}
0.862 (+/-0.024) for {'kernel': 'rbf', 'C': 1000, 'gamma': 1.0}
0.945 (+/-0.042) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.1}
0.948 (+/-0.028) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.01}
0.952 (+/-0.028) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.955 (+/-0.033) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.862 (+/-0.024) for {'kernel': 'rbf', 'C': 10000, 'gamma': 1.0}
0.945 (+/-0.042) for {'kernel': 'rbf', 'C': 10000, 'gamma': 0.1}
0.948 (+/-0.028) for {'kernel': 'rbf', 'C': 10000, 'gamma': 0.01}
0.945 (+/-0.035) for {'kernel': 'rbf', 'C': 10000, 'gamma': 0.001}
0.949 (+/-0.040) for {'kernel': 'rbf', 'C': 10000, 'gamma': 0.0001}
0.947 (+/-0.039) for {'kernel': 'linear', 'C': 1}
0.938 (+/-0.036) for {'kernel': 'linear', 'C': 10}
0.940 (+/-0.034) for {'kernel': 'linear', 'C': 50}
0.937 (+/-0.036) for {'kernel': 'linear', 'C': 100}
0.934 (+/-0.044) for {'kernel': 'linear', 'C': 500}
0.935 (+/-0.045) for {'kernel': 'linear', 'C': 1000}
0.933 (+/-0.045) for {'kernel': 'linear', 'C': 10000}
```

Figure J.1: Output of the grid search when testing hyperparameter values as shown in Table 4.4.